Multimodal Approach for Big Data Analytics and Applications

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Gautam Pal

September 2021

Abbreviation

Abbreviation	Expansion
ACD	Automated Cloud Deployment
AD	Active Directory
AUC	Area Under The Curve
AWS	Amazon Cloud Services
BLIP	Business Logic Integration Platforms
CAP	Consistency, Availability, and Partition Tolerance
CDH	Cloudera Distribution of Hadoop
CEN	Collaborative Ensemble Framework
CNN	Convolutional Neural Network
CTR	Click Through Rate
DDL	Data Definition Language
DNS	Domain name System
DSE	Datastax Enterprise
DStream	Discretized Stream
ECG	Electrocardiogram
ELLA	Efficient Lifelong Machine Learning
EM	Expectation-Maximization
FHV	For-Hire Vehicle
GDC	Genomic Data Commons
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Units
HCC	Human-centered computing
HDFS	Haddop Distributed File System
HDFS	Hadoop Distributed File System
HDS	Historical Data Store
HR	Heart Rate
IE	Information Extraction
IFBRS	Implicit Feedback Based Recommender System
ISR	In Sync Replica

IaaS	Infrastructure as a service
JDBC	Java Database Connectivity
KB	Knowledge Base
KM	Knowledge Miner
LA	Lambda Architecture
LDA	Latent Dirichlet Allocation
LIML	Lifelong Incremental Machine Learning
LML	Lifelong Machine Learning
LSC	Lifelong Sentiment Classification
LSTM	Long Short-Term Memory
MACDM	Multi-agent Coalitions Decision Making
MAF	Multimodal Analytics Framework
MALA	Multi-agent Lambda Architecture
MAP	Multimodal Analytics Platform
MAS	Multi-agent System
MCGDM	Multi Criteria Group Decision-Making
NB	Naïve Bayesian
NBP	Next Best Product Analysis
NER	Named Entity Recognition
NLP	Natural Language Processing
NMF	Non-negative Matrix Factorisation
NS	Neutrosophic set
OLAP	Online Analytical Processing
OSF	Open Science Framework
POS	Parts of Speech
PR	Purchase Rate
RCA	Root Cause Analysis
RDBMS	Relational Database Management System
RDF	Random Decision Forest
RDF	Resource Description Framework
RDQL	Resource Description Framework Query Language
RMSE	Root Mean Square Error
RNN	Recurrent Neural Networks
S3	Simple Storage Service
SGD	Stochastic Gradient Descent
SLC	Lifelong Sentiment Classification
SP	Stream Processor
SPL	Search Processing Language
SQL	Structured Query Language
SU-LML	Semi-Unsupervised Learning
SVC	Support Vector Classifier

SVD	Truncated Singular Value Decomposition
TF-IDF	Term Frequency-Inverse Document Frequency
USI	User Satisfaction Index
VADER	Valence Aware Dictionary and sEntiment Reasoner
XOS	Overall Surviving
YOLO	You Only Look Once

Preface

This thesis is primarily my own work. The sources of other materials are identified.

Abstract

The thesis presents multimodal conceptual frameworks and their applications in improving the robustness and the performance of big data analytics through cross-modal interaction or integration. A joint interpretation of several knowledge renderings such as stream, batch, linguistics, visuals and metadata creates a unified view that can provide a more accurate and holistic approach to data analytics compared to a single standalone knowledge base. Novel approaches in the thesis involve integrating multimodal framework with state-of-the-art computational models for big data, cloud computing, natural language processing, image processing, video processing, and contextual metadata. The integration of these disparate fields has the potential to improve computational tools and techniques dramatically. *Thus, the contributions place multimodality at the forefront of big data analytics, the research aims at mapping and understanding multimodal correspondence between different modalities.*

The primary contribution of the thesis is the Multimodal Analytics Framework (MAF), a collaborative ensemble framework for *stream* and *batch* processing along with cues from multiple input modalities like language, visuals and metadata to combine benefits from both *low-latency* and *high-throughput*. The framework is a *five-step* process:

Data ingestion. As a first step towards Big Data analytics, a high velocity, fault-tolerant streaming data acquisition pipeline is proposed through a distributed big data setup, followed by mining and searching patterns in it while data is still *in transit*. The data ingestion methods are demonstrated using Hadoop ecosystem tools like Kafka and Flume as sample implementations.

Decision making on the ingested data to use the best-fit tools and methods. In Big Data Analytics, the primary challenges often remain in processing heterogeneous data pools with a one-method-fits all approach. The research introduces a decision-making system to select the best-fit solutions for the incoming data stream. This is the *second step* towards building a data processing pipeline presented in the thesis. The decision-making system introduces a *Fuzzy Graph* based method to provide real-time and offline decision-making.

Lifelong incremental machine learning. In the third step, the thesis describes a *Lifelong Learning* model at the processing layer of the analytical pipeline, following the data acquisition and decision making at *step two* for *downstream processing*. Lifelong learning iteratively increments the training model using a proposed *Multi-agent Lambda Architecture (MALA)*, a collaborative ensemble architecture between the stream and batch data. As part of the proposed MAF, MALA

is one of the primary contributions of the research. The work introduces a general-purpose and comprehensive approach in hybrid learning of batch and stream processing to achieve lifelong learning objectives.

Improving machine learning results through ensemble learning. As an extension of the *Lifelong Learning* model, the thesis proposes a *boosting based Ensemble method* as the fourth step of the framework, improving lifelong learning results by reducing the *learning error* in each iteration of a streaming window. The strategy is to incrementally *boost the learning accuracy* on each iterating mini-batch, enabling the model to accumulate knowledge faster. The base learners adapt more quickly in smaller intervals of a sliding window, improving the machine learning accuracy rate by countering the *concept drift*.

Cross-modal integration between text, image, video and metadata for more comprehensive data coverage than a text-only dataset. The final contribution of this thesis is a new *multimodal method* where *three* different modalities: text, visuals (image and video) and metadata, are intertwined along with real-time and batch data for more comprehensive input data coverage than text-only data. The model is validated through a detailed case study on the contemporary and relevant topic of the COVID-19 pandemic. While the remainder of the thesis deals with text-only input, the COVID-19 dataset analyzes both textual and visual information in integration.

Post completion of this research work, as an extension to the current framework, multimodal machine learning is investigated as a future research direction.

Acknowledgements

In 2017, when I took a dramatic decision to start a PhD, I was then working in the leading global software companies for over 12 years. In the end, it took a strong conviction indeed to leave a mid-career, well-paid, and well-settled multinational company job, leaving behind family and to a take perilous lonesome journey to uncertainty. I would like to acknowledge the funding support and job assurance I received from my previous employer Accenture. Without Managing Director *Anupama Nityanand* and my Manager *Rahul Ghali*'s initial assurance of possible re-joining opportunity to Accenture, I never had enough courage to start a new career. I owe my previous colleagues in Accenture to get me started with Big Data back in 2012.

My parents, *Bijay, Sulekha Pal*, my wife *Banani Ghosh* endured financial difficulties for the period due to my decision but stood by to let me chase my dream of a PhD.

I would like to thank my supervisors over the years. Thank you, *Professor Gangmin Li*; it was extremely rewarding to listen and learn from your extensive experience. I was always enriched through your patience, motivation, kick-starting my thesis journey and helping me to acclimatize the academic environment from the industry. The different perspective to my research that you brought has been valuable. I could not have imagined having a better advisor and mentor for my PhD study. The lecture and lead Teaching Assistant opportunity that you entrusted upon me remained my best-cherished memories in my university life.

I am sincerely grateful to *Professor Katie Atkinson*. You always had all your time to review the thesis despite you being so much busy at such an esteemed position as a Dean of the cluster! I very much appreciate the time and effort you put into guiding me through my studies and providing valuable feedback whenever possible. I was overwhelmed with gratitude for your help to extend my stay at Liverpool, which in the end proved pivotal to finish this thesis and land a job at the University of Liverpool.

Thanks to advisors *Professor Yannis Goulermas* and *Professor Owen Liu* for your critical comments improving the thesis quality. I thank my fellow PhD friend for the stimulating discussions, the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

As I am putting the finishing touches on the thesis, the world has come to a standstill due to the pandemic. The final strides to the touchline of the thesis were completed outside the regular office walls. But still, there remained few bright spots in the midst of the gripping darkness. I started

working in my new job at the University of Liverpool as a *Research Associate*. Thanks to my new supervisor *Professor Kay O'Halloran* who has put me as the Lead Researcher role in the team.

May the new dawn soon arrive. And we will never walk alone.

This work is dedicated to all the animals suffering and constantly adapting to a changing face of the earth and endless cruelty by human beings. This world is as much theirs as it is ours. When I am feeling low, overwhelmed, or tired, they inspire me to hold till the end so that I can help make this world a little better place for all the animals. Thanks to all of them for inspiring me to become something greater than what I am.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The primary objective of this research is a joint interpretation of multiple knowledge representations to create a unified view that produces a more accurate and comprehensive depiction of data compared to a single standalone knowledge base. In big data analytics, the majority of the tools and methods are devoted to a single modality. The grand challenges of this research are to integrate or interact with all possible combinations of stream and batch data along with text, image, video and metadata. As a step forward in this direction, the thesis introduces Multimodal Analytics Framework (MAF) constituting of *three* main components: data ingestion, Multi-criteria Decision Making (MCDM) and a Multi-agent Lambda Architecture (MALA). The MAF is subsequently implemented in an easy-to-use web-based platform in a public cloud called Multimodal Analytics Platform (MAP).

Through the period of the last two decades, there has been significant progress in machine learning, big data, Natural Language Processing, computer vision algorithms and frameworks. However, much less emphasis was put on how these methods and algorithms can be integrated to train over an extended period of time to *incrementally* become more knowledgeable through knowledge retainment and transfer. Multimodal learning and prediction involve embedding of multi-sensory data that humans perceive in the form of reading, visualising and hearing within. Software models, despite recent advancements in machine learning and big data technologies, an-

alytical goals significantly fall behind a manual understanding of cognitive jobs in an aggregation of text, image, video and metadata. Consequently, in reality, the state-of-the-art computer models are still *monomodal* specialists, leaving the scope for humans to logically interpret joint modalities in aggregation. The efficiency of the state-of-the-art predictive analysis frameworks is limited within forecasting without adequate reasoning and root cause analysis [63, 231]. Apache Hadoop is the de facto standard for *batch-processing* systems used to provide high throughput, comprehensive, and more accurate views than streaming algorithms on a large historical dataset in terms of machine learning results and analytical output. However, large scale distributed systems such as Hadoop suffer from a number of challenges like high latency, increased storage and cluster requirements [61, 251]. The batch jobs tend to work on the entire data pool, resulting in a long completion time (high latency) but at a reasonably greater speed (high throughput) than the stream jobs. Batch job uses Hadoop Distributed File System *HDFS* as a *distributed storage* and *MapReduce* as a *processing* mechanism that involves large cluster requirements with several physical nodes. In contrast, stream processing frameworks like Apache Flink, Apache Storm, Apache Beam provide quick turnaround time, trading off with *imprecise* results and may suffer from *cold-start* situations [119, 150] (detailed in Chapter 3). With a trade-off between highly accurate batch and low latency Online Analytical Processing (OLAP), the most efficient data analytics are able to work effectively for both *low-latency* and *high-accuracy* processing requirements. Thus, a collaborative *ensemble* environment makes it easier to tap into the power of both the batch and real-time analytics, explore data at scale, incrementally train machine learning models, and visualise insights.

The main reason for the aforementioned research gaps is that the different modalities often require convergence of very different scientific fields. However, multimodal analysis and data mining performance, especially in terms of robustness, can be significantly improved by combining multiple modalities in the way of integration or collaboration. Thus, the critical research gaps, as mentioned in the previous paragraph, require to development of a *Collaborative Ensemble Framework (CEN)* with the capability of Lifelong Machine Learning for stream and batch data to combine benefits from both. The proposed Multimodal Analytics Framework (MAF) provides the

optimal fusion of disparate streams from multiple modalities and data sources. MAF minimises the manual interpretations of the combined modalities in a way that is capable of: (i) efficient information extraction, ingestion, and cleansing; (ii) effective integration of data sources from the stream, batch, text, visuals (image, video) and metadata in a cross-media interaction scheme; (iii) leveraging the latest technologies in data science, big data and NLP; and (iv) easy to use intuitive web-based interface, making technological advancements in multimodal analysis accessible to the non-data scientists (social scientists, linguists, etc.).

The proposed framework enables both *high-throughput* on batch-oriented tasks along with *low-latency* real-time responses in an environment characterized by *data heterogeneity*. The MAF enables the graceful interaction of stream data along with historical batch data at the initiation and thereafter in an autonomous way. The collaborative framework between batch and real-time leads to developing an *Incremental Lifelong Machine Learning (ILML)*, which imitates humans in their never-ending learning experience.

In addition, there have been research gaps between the existing theoretical advancement and practical implementations, with respect to the research questions set out in Section 1.2 that expose the need for investigation and provide a solution at the implementation level. The thesis also offers a solution to the key research questions through a careful study of the current state-of-the-art in terms of theoretical contribution and implementing strategies for reproduction in practice.

1.1 Motivation

Using the past acquired knowledge to improve future learning leads to the next generation of machine learning. Lifelong learning mimics *incremental* and *continuous learning* experiences of human life. Training a large volume of a historical dataset requires large clusters and expensive infrastructure to make the long-running one-shot batch jobs unrealistic on a big data pile. Batch jobs are not agile enough to adapt quickly with each new wave of ever arriving datasets. Hadoop and its de facto MapReduce framework make the learning process wait until a full set of data are collected from the data sources [91,140]. Analytical outcomes are not published until long-running

batch jobs are completed for hours or even days.

On the contrary, streaming learning enables models to get trained and updated after each passing mini-batch of a window. The model can catch up to the newest trends faster and with a much smaller cluster size, reducing the infrastructure overhead significantly. Nevertheless, to begin with, in the analytical or learning process, streaming learning can get overwhelmed with a large set of the historical data pool. To address the *responsiveness* issue with the traditional batch model, this research introduces a mix-model approach through a big data lambda architecture that considers batch and stream as two collaborating multi-agent systems.

To start with an analytical or learning process, the batch system stores past learning into the HDFS for later use by a stream engine. The streaming system then updates the model incrementally. The streaming model initialises itself with *saved learning* from the batch by loading the trained model from HDFS into Apache Spark *Discretized* Stream (DStream). A configurable *mini-batch* of the time window of a few hours re-trains the model and, at the same time, predicts continually on test data. The updated model through the stream engine is persisted and replicated into HDFS at a periodic interval to run any ad-hoc batch query. Additional static data can be merged into previously-stored HDFS data by a stream processor, and the mini-batch can pick and continue from thereon. The proposed stream-batch hybrid model is implemented using Expectation-Maximization (EM) algorithm with Gaussian Mixture Model (GMM) (Chapter 3) and clustering algorithm (Chapter 5).

Nevertheless, analysis approaches of batch and real-time are predominantly language-based and lack theoretically well-founded methods for addressing relevant meanings that emerge from juxtapositions of visual messages, such as images and videos. Such juxtapositions are increasingly seen as decisive in analytical outcomes. As a consequence, existing big-data approaches are insufficient for understanding the impact of such multimodal dataset and their effects within many critical analytical insights.

1.2 Research Questions

Traditional big data applications have a hard choice between correctness or faster response in realising analytics and machine learning tasks. However, modern complex systems often require the synthesis of both features in the same system. Also, the next-generation systems can not just rely on only *textual* data without considering associated visual components (image and video) into the analytical model. Therefore, an alternative approach requires integrating the *multiple* different methodologies to achieve the overall higher objective of an efficient machine learning system. The potential research gaps lead to developing a novel end-to-end analytical system from data generation to machine learning prediction, placing *multimodality* at the forefront of research. The proposed *ensemble* system will address the key question:

- **Multimodal join representation**. How to provide a joint interpretation of several knowledge representations in terms of multiple modalities such as *stream, batch, linguistics, visuals, metadata*, and create a unified view that produces a more accurate and comprehensive representation of data compared to a single standalone knowledge base?

The sub-questions making up the larger question are as follows:

1. How to develop a high velocity, fault-tolerant streaming *data acquisition pipeline* for the downstream processing as the first layer in the MAF?

2. **A Multi-agent System (MAS) decision-making model:** What is the technique to extract appropriate domain knowledge at the data ingestion layer and dynamically select the suitable problem solver(s) for downstream analytics and machine learning, based on the multi-agent decision-making model?

3. **A multimodal collaborative model between batch and stream processing:** How does a big data model consolidate low-latency real-time framework with high throughput Hadoop batch framework over a large distributed setup? In particular, real-time and batch processing engines are two collaborative agents of an autonomous multi-agent system.

4. **A Lifelong Learning Model**: How does a machine learning model incrementally updates its knowledge base continuously and lifelong through a collaborative framework between batch and real-time?

5. **An Ensemble Learning Model:** What are the methods for streaming big data *Ensemble Learning* to improve prediction outcomes over the period?

6. **A multimodal model to integrate *three modalities***: How can *three modalities*: text, visuals (image and video), and textual metadata be integrated for analytical tasks such as multimodal classification, Topic Modeling, entity correlation, sentiment classification and clustering?

On a high level, The research question and sub-questions correlates to the end-to-end big data analytics framework from data ingestion to machine learning that is laid out in the thesis. Sub-question 1 and 2 relates to Chapters 3 and 4. Chapters 5, 6 addresses Sub-question 3. Chapter 5 also deals with Sub-question 4 and Chapter 6 covers Sub-question 5. Sub-question 6 is addressed in Chapter 7.

The following section summarises the key contribution and the thesis outline in different chapters with respect to the research questions set out. The contributions are validated along with different case studies in *five* major domains of big data: e-commerce, healthcare, transportation, social and news media analysis as discussed in Section 1.4.1 which also describes the correlation between the research questions and an application area (case study).

1.3 Summary of Contribution

The research introduces a novel big data framework for simultaneously mixed processing architecture between the batch and the stream, leading to the development of a lifelong learning system. The proposed MAF provides the optimised fusion of disparate streams from multiple input modalities such as text, visuals and metadata. Outcomes include a critique of existing methods for gathering data and the algorithms currently control the flow of information for multimodal ensemble

learning methods. The framework is a *five steps* working procedure. The key contributions of this research are as described below:

1.3.1 A Collaborative Ensemble Framework for Big Data

Ensemble learning helps improve machine learning results by combining several machine learning methods. The approach produces better predictive performance compared to a single model providing synergy between various scientific fields and several research methodologies. Proposed Ensemble methods are meta-algorithms that combine several machine learning cues into one predictive model in order to decrease variance (bagging) and bias (boosting) or improve predictions (stacking) [69, 294]. For instance, to predict future weather from past time-series data, several algorithms can be employed. A *meta-algorithm* takes a weighted average of each underlying method to provide an *overall* estimate.

The work proposes a novel ensemble framework for big data analytics through the proposed MALA. MALA is a collaborative ensemble framework for stream and batch data. In the MALA framework, a Knowledge Miner (KM) consolidates the past learning with recent stream updates into a *Knowledge Base (KB)*, transforming the model into a *Lifelong Learning System*. Initialising with the batch processing results as an offset, the framework successively applies: (i) a streaming clustering procedure to group the target data points, (ii) followed by a *Random Decision Forest Regressor* and *classification* algorithm to provide reasoning through dynamic root cause analysis, comparing between groups of a disparate dataset and performs forecasting. The model's real-time and batch agents use Spark machine learning libraries. Proposed algorithms are scalable using the distributed in-memory processing APIs of Spark MLlib. Spark ML algorithms provide a standard implementation for frequently used learning methods like Clustering, Classification, Collaborative Filtering, and Regression. Spark ML [39, 172] pipeline tools are used for evaluation and fine-tuning an existing implementation to benchmark many results presented through the thesis.

The batch and real-time components of the MALA are two autonomous agents that collaborate at need. MALA introduces novel methods that optimise big data analytics by exploiting the inter-

action of low-latency stream processing with comprehensive batch processing across textual and visual information.

Lambda Architecture (LA) [133] is a new direction towards the next generation of big data analytics, which started to witness a wide industry adaptation in recent years. However, research efforts towards LA are still very limited. This work introduces a general-purpose and comprehensive approach in combination with big data and Multi-agent Lambda Architecture that is designed for hybrid learning of batch and stream data processing to realise the lifelong learning objectives.

The MALA produces superior predictive performance compared to state of the art consisting of *only single model*—batch or stream. The framework has a unique *Lifelong Learning* proposition to incrementally *boost* the knowledge learned in the past. The incremental learning approach has an advantage over the existing *one-shot* learning methods in terms of faster learning time, limited resource utilisation, and data availability. The proposed techniques are applied in supervised and unsupervised learning algorithms when training data is not fully available at the beginning and gradually arrives over time. The framework can initiate on a small set of sliding windows, removing any resource bottleneck when the full data size outfits the system bandwidth in terms of memory and disk.

1.3.2 Text-image-video Cross-modal Correspondence

As a *second* contribution, the research introduces new methods to join multiple modalities to carry out a prediction task. Novel approaches are proposed for multimodal classification, Topic Modeling, Entity Correlation, and Clustering to train and predict multiple modalities between language, visuals (image and video), along with metadata. The research outcome indicates that analysing datasets with multiple modalities such as language, metadata, and visual cues can increase prediction accuracy than traditional mono-modal systems using text-only data.

1.3.3 Data Acquisition Pipeline and Decision Making

The *third* contribution of the thesis is a high velocity, fault-tolerant streaming data acquisition pipeline in a distributed big data setup, mining, and searching patterns in transit while data is still not stored in a database. The thesis introduces MCDM, a graph-based method for an offline and online (real-time) decision-making system that selects the best-fit methods for the incoming data stream. The approach solves the data *variety* problem of big data by applying different methods for each dataset in a *heterogeneous* data environment consisting of multiple data formats and schema.

1.3.4 Streaming Clustering through Dimension Reduction

As the *fourth* contribution of the thesis, is an *incremental learning method* through in-memory *Streaming Clustering* that uses *Dimension Reduction* to reduce training time. This task aims to achieve clusters of similar data profiles and isolate the specific clusters for further investigations. The proposed method uses streaming K-means techniques extending the standard K-means algorithm [259, 293]. The basic idea for a streaming version of K-means clustering is to divide the data stream into mini-batch windows and to incorporate knowledge learned in the previous window into the subsequent ones. Hence, in streaming K-means clustering, the model is updated with each rolling window based on a combination of cluster centres computed from the preceding mini-batches and the current mini-batch. The method further enables the merging of a large static historical data pool with the latest and most updated streaming model. The work presents a unique dimension reduction method of feature vectors enabling quick re-training for clustering and Random Decision Forest through the MALA architecture. With this approach, the batch model creates the offset (or initial point). On each streaming window of the dataset, the streaming model only needs to learn the updated features or corrections during the re-training process.

1.3.5 Random Decision Forest-based Forecasting and Root Cause Analysis

As a *fifth* contribution, a Random Decision Forest Regressor graph further drills down into each anomalous cluster and provides a comprehensive root cause analysis and prediction for future

items. The proposed method produces the Decision Tree graph representation in JSON formatted data as well as in a tree structure. Several optimisations are proposed for the Random Decision Forest model, including data sampling, important feature selection, and optimal Decision Tree hyperparameters to improve the training time through reduced data volume without affecting the overall accuracy.

1.3.6 Analyzing Geospatial Time Series Data

Mining insights from the voluminous and ever-generating stores of geospatial-temporal big data carries a substantial challenge. As a *sixth* contribution, the proposed ensemble model is implemented to analyse data linked to location and time to demonstrate the tools and technologies proposed in terms of collaborative ensemble framework for big data. Data comprises satellite and aerial images, global-scale data, geo-referenced IoT/sensor systems, and big data events ingested on an array of computing platforms. Geospatial data analysis is a case study for the thesis considering the amount of data generated for batch processing as well as time-series data for real-time simultaneous processing.

1.3.7 Fault-tolerant Distributed Big Data Infrastructure

The experiments presented in this thesis are conducted at large clusters hosted on public cloud environments. A fault-tolerant distributed big data infrastructure is presented to support the large storage and processing requirement. The experiments are set up on the Amazon Cloud Services (AWS) for Cassandra, Kafka, Spark, and Hadoop installations. The Apace storm is installed in the Microsoft Azure cloud, and the Multimodal Analytics Platform (MAP) is hosted on Google Cloud. Therefore, the infrastructure setup on which the experiments are carried out in the thesis provided an alternative approach that involves integrating multimodal framework with state of the art computational models for big data, cloud computing, natural language processing, image processing, video processing, and contextual metadata.

In summary, the following are the key novel aspects of the proposed MAF:

1. Integrating up to *five* modalities along with language, metadata, and visual cues to improve the prediction accuracies compared to traditional mono-modal or bi-modal approaches with text and image only datasets.

2. An ensemble system can simultaneously perform both batch and streaming jobs but with similar efficiency compared to state-of-the-art *batch only* or *stream only* jobs operating independently.

3. Collaborative framework between batch and real-time processing provides a novel way to develop a Lifelong Machine Learning though *incremental* approach. The learning model is trained over an extended period to incrementally become more efficient through knowledge retainment and transfer compared to one-shot training on a large data pool.

4. A novel Multi-Agent Decision Making (MCDM) model offers a solution to data *variety* problem of big data by applying different methods for each dataset in an environment with a *heterogeneous* data pool. Therefore, the MCDM is efficient than the traditional one-method fits-all-data approach.

5. Introduces new methods for real-time ingestion data, in-flight processing and distributed storage to analyse streams of data.

1.4 Thesis Outline

This thesis contains *eight* chapters. An end-to-end data flow pipeline represented through Figure 1.1 describes the work presented in Chapter 3 to Chapter 7, focusing on the original contributions that identify new research directions. Chapter 1 and Chapter 2 are for *Introduction* and *Literature Review*. Chapter 8 summarises the thesis and sets out a future direction for the research.

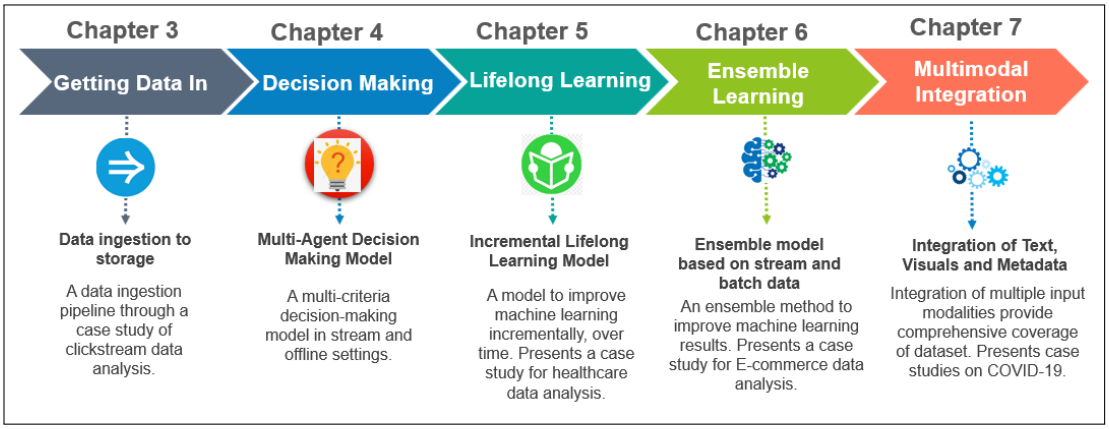


Figure 1.1: Thesis outline from Chapter 3 to Chapter 7 representing an end-to-end data flow pipeline.

As depicted in Figure 1.1, data ingestion is the first layer in the proposed Multimodal Analytics Framework (MAF) presented in Chapter 3. Once data is ingested into the framework, a decision-making system selects the best-fit methods for each incoming data stream as described in Chapter 4. As the next component in the analytical pipeline, Chapter 5 presents a *Lifelong Learning* model at the processing layer after data is acquired, and a set of tools are selected for the *downstream processing*. Chapter 6 extends the previous chapter by presenting a *boosting based ensemble method* that improves lifelong learning results. The concluding Chapter 7 integrates non-textual input along with textual content across real-time and batch formats described in the previous chapter for comprehensive coverage of a dataset.

The thesis presents research on the topic of big data *streaming and hybrid learning* that leads to developing a Lifelong Machine Learning (LML) system. Established frameworks for big data eco-systems, especially stream processing tools, Natural Language Processing and multi-agent systems, are used throughout the thesis chapters. A generic collaborative model between batch and stream is developed on top of a standard Lambda Architecture. The research achieved integrating of up to 4 modalities: language, image, video and textual metadata. In the proposed Multimodal Analytics Framework (MAF), stream and batch modules of big data are two collaborating agents to help derive greater business insights that need both stream and batch processing results at the same

time. A cloud-based Multimodal Analysis Platform (MAP) is developed for integrating disparate streams from multiple modalities and data sources.

To summarise, this thesis is structured into seven chapters outlined as follows:

Chapter 2 presents a literature review on the topics related to Lambda and Kappa Architecture, Lifelong Learning models, multi-agent decision-making paradigms, and multimodal approaches to integrate language, visuals and metadata. The relevant background is provided for big data ingestion tools like Apache Kafka, Flume, processing engines such as Storm and Adobe Analytics, and datastores like Casandra and HBase. Related work for the case studies in the thesis is described, e.g. recommender system, spatiotemporal analysis and social media analysis.

Chapter 3 presents methods to ingest data into big data storage through a distributed fault-tolerant manner. It is worth noting that the bottleneck of many data science solutions is the data ingestion component. Therefore, Novel techniques are demonstrated for stream data ingestion and storage through hosts of Big Data tools like Kafka, Flume, Spark, and Cassandra. The model ingests data in all shapes and sizes. Data is processed *in-transit* before it is stored into a Big Data storage.

Chapter 4 presents a multi-agent decision-making system to choose the right set of tools for each data stream. A multi-agent graph-based decision maker component is placed at the gateway to the data pipeline for downstream processing. The decision-maker selects best-fit methods and algorithms at every streaming window of ingested data, as discussed in the previous chapter.

Chapter 5 presents a *Lifelong Learning* model-based ensemble framework between batch and real-time components. The batch system saves its learning to the Hadoop Distributed File System (HDFS) for later use by a stream engine. The streaming system can then update the model incrementally on the batch result offset. The model gets re-trained and, at the same time, predicts continually on test data, making it a *Lifelong Learning System*. A lifelong learning paradigm is proposed to solve the sentiment classification problem under an unsupervised learning setting with previous knowledge.

Chapter 6 presents an *Ensemble Learning* method to improve lifelong machine learning re-

sults. Novel *boosting* methods are introduced for streaming big data machine learning in collaboration with batch data. Unique dimension reduction methods are presented for the feature vector to reduce the model training time significantly.

Chapter 7 further improves the *Ensemble Learning method* by integrating up to *four* data input modalities: language, visuals (image and video) along with metadata for comprehensive coverage of data. The chapter introduces new multimodal classification methods, Topic Modeling, Entity Correlation and Clustering, to train and predict. An integrated semi-automated, cloud-based open and extensible prototype of the multimodal system is developed as part of this research.

Chapter 8 presents the thesis conclusions and identifies possible future research paths.

1.4.1 Case Studies

The proposed framework's application areas are based on the case studies dealing with *large volume of time-sensitive data* that require *both batch and real-time solutions*. Case studies are implemented through various *e-commerce applications* with data source from e-commerce major *Amazon*, the largest Indian e-commerce portal *Flipkart* and the non-commercial, movie rating repository *movielens*. A case study is presented on *health care systems* with prostate cancer patients pathological dataset. A case study validates the proposed framework with a spatiotemporal dataset from *New York taxi app* data. Finally, *three* separate case studies are introduced using the COVID-19 dataset across social media (Twitter) and *five* U.K. newspapers. The case studies are selected based on the *five* major application areas of big data: e-commerce, healthcare, transportation, social and news media analysis due to intrinsic heterogeneity, time-sensitive characteristics and a large volume of the datasets. The general evolution criteria for the case studies are to prove or disprove the efficiency of multimodal analytics compared to mono-modal data processing.

E-Commerce Applications

The case study (in Chapter 6) evaluates the contributions laid out in the main Research Question 1.2 and the following Sub-questions 3, 4 and 5 . The case study exploits both *low latency tasks*

such as *recently viewed products* and outlier detection, as well as *high throughput tasks* such as *Recommender Systems*. The case studies are based on the MALA that builds the *recently viewed products*, forecasts network traffic load, analyses user browsing pattern etc. in a real-time setting. Using the batch and real-time integration, the model simultaneously can analyse *personalised recommendations*, and *motifs of users* described as follows:

Recommender System

Using the proposed synthesis in Chapter 5 of real-time, batch data and lifelong incremental learning, novel methods are introduced for an *implicit feedback* based recommender system. The key features of the application are:

1. Implicit Feedback Based Recommender System (IFBRS) uses users' implicit feedback in the form of item views. The model interprets item-to-item collaboration in one browsing session by each user compared to the traditional *user to user similarity approach*. IFBRS extracts users' location of browsing as a latent factor for increasing the relevancy of recommendations. Context-aware recommendation tends to be more accurate.

2. The model redefines the item-to-item relationship by applying higher relevance to the current trend than the historical data.

3. A method integrates results from *Collaborative Filtering* with *Content-based Filtering* depending on the item similarity to improve recommendation accuracy and resolve sparsity problem.

4. Based on the above studies, the work presents a novel architecture of an end-to-end recommender system with a host of online and offline big data eco-system tools and their correlation as a multimodal interactional model. The proposed methods have potential beyond the recommender system for the e-commerce domain and can be extended to other domains such as recommending financial or telecommunication products to users.

Other E-Commerce Functionalities

A list of functionalities are demonstrated in Chapters 3, 6 that can be performed *simultaneously* with Recommender System at the streaming layer of MALA as follows:

1. The case study presents novel analytical techniques for in-flight stream data using real-world instances of clickstream data to unleash key customer journeys and pattern mining using n-grams and Student T-Test, which distinguishes between regular patterns and special sequences.

2. Clustering clickstream sequences help generate customer segments and build personalised recommendations. The custom-built model integrates the batch and real-time data pool. First, the initial cluster is built in the batch setting. Thereafter, at a streaming setting, a variation of the *Expectation-Maximisation* algorithm [178] with *Gaussian Mixture Model* maps each user click events to one of the clusters at every streaming window interval. A dimension reduction technique reduces training time significantly.

3. Presents a predictive analytics method to forecast *network traffic load* on e-commerce portals.

4. Ingested clickstream is cleansed and processed in real-time to serve the UI transactions and display the recently viewed items to enhance the overall shopping experience. A list of *recently viewed items* determines the final order of recommended products using a weighted average between historical batch data and real-time data. The real-time setting simultaneously can detect the outliers in network bandwidth usage through a proposed hybrid clustering algorithm.

5. A generic stream processing model is built on the Apache *Storm topology* serving near-real-time responses.

Predict Emerging Situations During COVID-19 Pandemic

The case study laid out in Chapter 7 evaluates the contributions of the main Research Question 1.2 and the specific Sub-question 6. How to predict social media growth in relation to the observed news media volume is the focus of this case study. Detailed analysis is carried out on the persistence of long term trend dynamics on COVID-19 in conventional print and social media (Twitter) and provide a mathematical basis for trend formulation, surge, persistence, and eventual decline. The case study uses MAF for news and social media that infer past data to provide early indications of the impacts due to epidemic spread, such as new illnesses, recoveries, deaths, and job losses purely based on social media conversations. The proposed MAF effectively models the amplification of economic, social, and public health and related emerging issues posted on Twitter. Analysing the social media conversations, the MAF discovers language, visuals, and metadata in real-time through complex web searches; indexes annotates and aggregates them to create interactive reports.

Healthcare Data Analysis

A case study from healthcare data is developed to validate the proposed MALA based on a large volume of a high dimensional dataset, evaluating the contribution relating to the key Research Question 1.2 and Sub-questions 3, 4 and 2. The case study is described in Chapter 5. The National Cancer Institute's Genomic Data Commons (GDC) releases the unified data repository for the cancer research community to support personalised medicine. The repository contains over 32000 patient cases, including clinical data, treatment data, biopsy results, gene expression data, as well as a whole host of other information [1]. Thus, allowing accessibility to researchers who want to uncover novel biomarkers and find a correlation between genes and survival.

Personalised cancer medicine is customised to each individual patient's need for chemotherapy or drugs based on patients' specific set of DNA or genes. A cancer patient's pathological tests like blood, DNA, urine, or tissue analysis provide a unique signature based on the DNA combinations.

The analysis aims to improve patients' overall survival rate with prostate cancer through personalised and targeted medications and achieves a therapeutic response based on the thousands of

genotype and phenotype parameters. Data are clustered based on the proposed stream-batch hybrid clustering method for patient profiling and detecting the outliers. Outliers are further analysed with a Decision Tree for root cause analysis to improve the Overall Surviving rate. The efficiency of the process gets better over time with the incremental Lifelong Learning approach.

New York Taxi Trip Dataset Analysis

The case study examines the New York taxi trip dataset in Chapter 6. The case study illustrates the efficacy of the proposed MALA on a *large volume of time-sensitive dataset*, further validating contribution towards the Research Question 3. Spatio-temporal queries perform data mining and visualisation tasks at the batch layer to analyse the city's mobility patterns such as traffic condition, fastest routes, weekend hotpots and day of the week taxi demand.

Benchmarking NoSQL Databases

The case study evaluates the contribution relating to the Research Question 2 in Chapter 4. The case study benchmarks *three* NoSQL databases: Cassandra, HBase, and MongoDB with respect to their overall against a set of criteria using the proposed algorithm.

1.5 Published Work

A summary of the publications (to date) relating to the work presented in this thesis is as follows:

Papers published

1. Kay L. O'Halloran, **Gautam Pal**, Minhao Jin (2021): Multimodal Approach to Analysing Big Social and News Media Data. In: Discourse, Context & Media, Elsevier, pages 100467, DOI: 10.1016/j.dcm.2021.100467 **(Impact Factor: 1.38)**. The publication relates to the multimodal integration techniques described in Chapter *7*.

2. **Gautam Pal**, Gangmin Li, Katie Atkinson: Lifelong Machine Learning and Root Cause Analysis for Large-Scale Cancer Patient Data (2020). In: Journal of Big Data, Springer,

pages 1-29, doi: 10.1186/s40537-019-0261-9. **(CiteScore 6.1, SNIP 2.501)**. The publication presents a lifelong learning framework described in Chapter *5*.

3. **Gautam Pal**, Gangmin Li, Katie Atkinson (2020): Managing Heterogeneous Data on a Big Data Platform: A Multi-Criteria Decision Making Model for Data-Intensive Science. In: BigComp 2020, pages 229-239, DOI: 10.1109/BigComp48618.2020.00-69. **(average acceptance ratio 23%)**. This publication presents an approach to solve the data *variety* problem of big data through an offline and online decision-making system described in Chapter *4*.

4. Hong, Xianbin and Pal, Gautam and Guan, Sheng-Uei and Wong, Prudence and Liu, Dawei and Man, Ka Lok and Huang, Xin (2020): Semi-Unsupervised Lifelong Learning for Sentiment Classification: Less Manual Data Annotation and More Self-Studying. In: 3rd High-Performance Computing and Cluster Technologies Conference, Guangzhou, China, Pages: 87-92 DOI: 10.1145/3341069.3342992. The publication presents a semi-unsupervised lifelong sentiment classification approach described in Chapter *5*.

5. **Gautam Pal**, Gangmin Li, Katie Atkinson (2019): Big Data Ingestion and Lifelong Learning Architecture. In: IEEE BigData 2018, Seattle, USA, pages: 5420-5423, doi: 10.1109/BigData.2018.8621859. **(average acceptance ratio 20%)**. The publication presents the proposed MALA to combine historical batch data with live streaming data to develop a lifelong learning system described in Chapters *5*, *6*.

6. **Gautam Pal**, Gangmin Li, Katie Atkinson (2018): Multi-agent Big Data Lambda Architecture Model for E-Commerce Analytics. In: Data, MDPI, pages: 58, DOI: 10.3390/data3040058. The publication presents big-data hybrid-data-processing lambda architecture, which consolidates low-latency real-time frameworks with high-throughput Hadoop-batch frameworks over a massively distributed setup described in Chapter *6*.

7. **Gautam Pal**, Gangmin Li, Katie Atkinson (2018): Top-n Recommender System Using Big

Data Item-to-Item Collaborative Filtering. In: International Conference on Internet Studies, Takamatsu, Japan. The publication presents the contextual item to item Collaborative Filtering described in Chapter *6*.

8. **Gautam Pal**, Gangmin Li, Katie Atkinson (2018): Big Data Real-Time Ingestion and Machine Learning. In: IEEE Second international conference on Data Stream Mining & Processing, Lviv, Ukraine. pages: 25-31, DOI: 10.1109/DSMP.2018.8478598. The publication presents machine learning approaches to analyse streams of data described in Chapter *3*.

9. **Gautam Pal**, Gangmin Li, Katie Atkinson (2018): Near Real-Time Big Data Stream Processing Platform Using Cassandra. In: IEEE 4th International Conference for Convergence in Technology, Mangalore, India, pages: 1-7, DOI: 10.1109/I2CT42659.2018.9058101. The publication presents near real-time data storage and processing approaches to analyse streams of data with respect to Cassandra NoSQL datastore described in Chapter *3*.

10. **Gautam Pal**, Gangmin Li, Katie Atkinson (2018): Big Data Real-Time Clickstream Data Ingestion Paradigm for E-Commerce Analytics. In: IEEE 4th International Conference for Convergence in Technology, Mangalore, India, pages: 1-5, DOI: 10.1109/I2CT42659.2018.9058112. The publication presents high velocity, fault-tolerant streaming data acquisition pipelines in a distributed setup described in Chapter *3*.

11. **Gautam Pal**, Gangmin Li, Katie Atkinson (2017): Multi-agent item to item contextual Big Data recommender system. In: International Journal of Design, Analysis, and Tools for Integrated Circuits and Systems, pages 58-59. The publication presents a recommendation system for non-authenticated (not logged in) user described in Chapter *6*.

12. **Gautam Pal**, Gangmin Li, Katie Atkinson (2017): A Multi-Agent Model for Big Data Analytics through Knowledge Graph. In: International Conference on Big Data Analytics and Business Intelligence (ICBDI), in Suzhou, China. The publication presents a novel way based on the Multi-agent Systems (MAS) and Knowledge Graph(KG) based approach to

derive appropriate domain knowledge and dynamically to find out a suitable problem solver described in Chapter *4*.

13. Gangmin Li, Minghuang Chi, **Gautam Pal** (2017): Expert CF: Sparse data matrix completion with artificial experts. In: International Journal of Design, Analysis & Tools for Integrated Circuits & Systems, pages 20-22. The publication presents the concept of *artificial experts* to overcome the matrix sparsity problem in the recommendation systems described in Chapter *6*.

Papers forthcoming/revised/submitted

14. **Gautam Pal**, Katie Atkinson, Gangmin Li (2021): Real-time User Clickstream Behavior Analysis Based on Apache Storm Streaming. In: Electronic Commerce Research, Springer (revised and resubmitted, **Impact factor 2.507**). The publication relates to-flight real-time data processing pipeline described in Chapter *3*.

15. **Gautam Pal**, Kay L. O'Halloran, Minhao Jin (2021): Persistence and Decay of Trends: Dynamics of News and Social Media as COVID-19 Emerged and Spreads. In: Appliable Studies of Linguistics, Beijing: Peking University Press (accepted, forthcoming). The publication relates to the trend dynamics of mainstream newspaper articles on the COVID-19 pandemic using multimodal integration described in Chapter *7*.

16. **Gautam Pal**, Kay L. O'Halloran, Minhao Jin (2021): Multimodal Approach to Predict Emerging Situations During COVID-19 Pandemic. In: Revista Signos, Pontifical Catholic University of Valparaiso Press, Chile (accepted, forthcoming, **Impact factor 0.61**). The publication presents the multimodal analysis of news and social media that infers past data to provide early indications of the impact of COVID-19 described in Chapter *7*.

17. **Gautam Pal**, Katie Atkinson, Gangmin Li (2021): An Efficient System Using Implicit Feedback and Lifelong Learning Approach to Improve Recommendation. The Journal of Supercomputing, Springer (revised and resubmitted, **Impact factor 2.4**). The publication presents

the techniques for contextual item-to-item Collaborative Filtering based Recommender System described in Chapter *6*.

18. Kay L. O'Halloran, **Gautam Pal**, Minhao Jin (2021): A multimodal big data approach to political branding. In: Vernon Press (submitted). The publication relates to the multimodal integration techniques described in Chapter *7*.

Chapter 2

Background and Literature Review

This chapter provides an in-depth analysis of the background and previous researches leading to this work. The literature review is divided into *seven sections* which are elaborated later in the thesis through Chapters *three* to *seven*. The chapter sets out with background for *capturing stream data* in big data analytics scenarios. A comprehensive analysis is provided on the current state of the art of the big data ingestion using popular frameworks like *Flume* and *Kafka* in the ingestion layer, *Storm* and *Spark* processing in the layer. Big data *Lambda Architecture* and its *v*ariant Kappa Architecture is described in the Section 2.3. Lambda Architecture is a general-purpose, fault-tolerant big data *hybrid* processing paradigm which is the base model for the frameworks defined int the Chapters 5, 6. A Multi-agent decision-making component is placed at the gateway of the proposed MALA architecture defined in 5. Existing research in the area of Multi-agent Coalitions Decision Making (MACDM) systems are elaborated in Section 2.4. A background of *Lifelong Incremental Machine Learning (LIML)* is given in Section 2.5, and the related background of a case study is presented in Section 2.6. Later in the thesis, Chapters 5 and 6 offer detailed discussion on the new methods for lifelong learning architecture and implementation through *recommendation engine*. Literature review on the current research efforts towards *geospatial, temporal data* is discussed as a *precursor* to the case study presented with the real-world data from the New York taxi app in Chapter 6. Chapter 7 presents an *Ensemble Learning* method by integrating up to *three*

input data modalities: language, visuals (image and video) along with metadata for comprehensive coverage of dataset. The methods are validated using a dataset from social and news media. Earlier works in the field of multimodal joint representation and existing social media analytics platforms are explained in the concluding part of the chapter (section 7).

2.1 Ingesting Stream Data

The first step in big data analytics is the data *ingestion* that allows getting data into the big data store. Big data *ingestion* accumulates data and transfers it to a system where it is persisted, analysed, and visualised. Data ingestion is the first step towards an efficient data strategy (generation to visualisation). Ingestion is as important as the processing itself. The following are the characteristics of a stream data ingestion platform:

- **Reliability**: Critical data streams should be delivered without any information loss.
- **High Throughput**: Should be able to handle large volumes of log or event data streams.
- **Low Latency**: Should be able to provide data with low latency for real-time applications.
- **Persistence**: Should persist data for a long duration to be consumed by batch systems such as Hadoop, which processes data periodically.
- **Scalability**: The cluster can scale horizontally as the stream data sources and sinks increase in number and volume.
- **Integration Points**: Should have easy integration with stream processing applications such as Spark, Storm, Samza, Kinesis, and Flink.

In this thesis, real-time streaming data platforms are presented with case studies for real-time user behaviour tracking in e-commerce websites, recommendation engines, geospatial data analysis, healthcare systems, and large scale big data infrastructure management as shown in Figure 2.1 [295, 299].

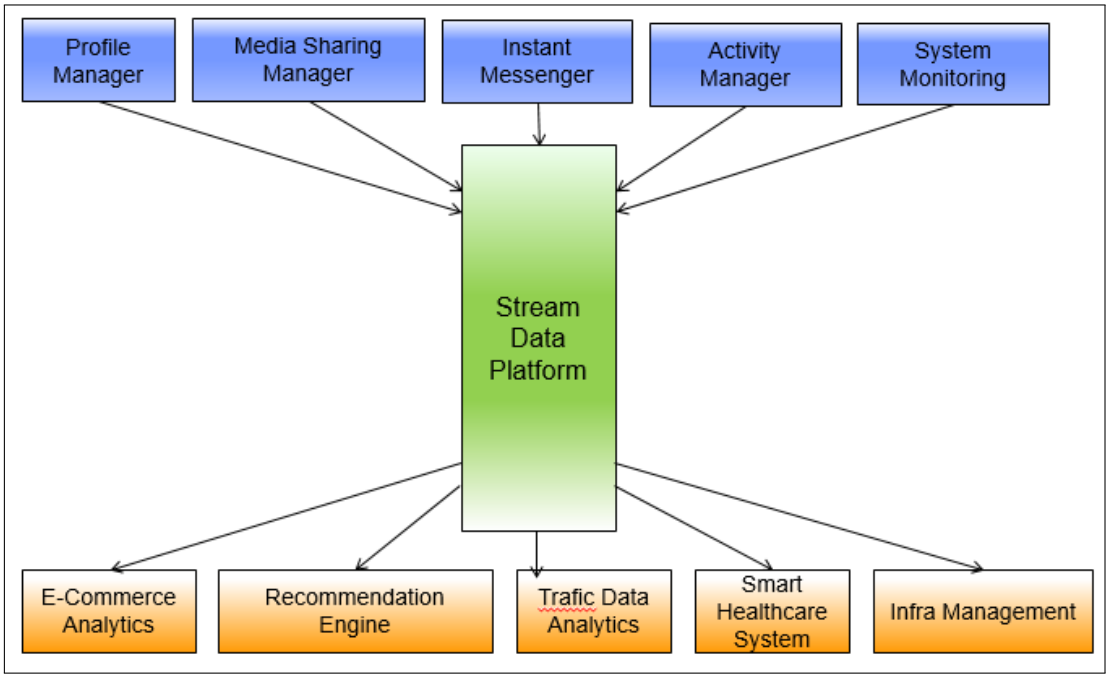


Figure 2.1: Data sources and application areas for a streaming platform as presented in this thesis.

2.1.1 Current Methods Ingesting and Analysing Stream Data

An efficient ingestion strategy involves extracting information from different data sources (as shown in Figure 2.2) and carrying it over a network in a fault-tolerant and distributed manner. For instance, *Flume* [49, 107, 108] uses *Netcat* or *Spoodir* abstractions to read data from sources like *web services* or a directory that is streamed with new data. See Figure 2.3 for a list of *abstractions* that Flume uses to read data from various sources. Ingested data is stored in a sink: either HDFS or a NoSQL database. Between source and sink, data can be stored into a *passive store* for a configurable amount of time or size when a sink is busy or unavailable for fault-tolerance. See Figure 2.4 for an end-to-end flow of data ingestion through Flume.

Apache Kafka [74, 84, 260] is a fast, scalable, durable and fault-tolerant publish-subscribe messaging system for data ingestion. It replaces traditional message brokers like Rabbit MQ, IBM-MQ because of higher throughput, reliability, and replication capabilities. It was originally

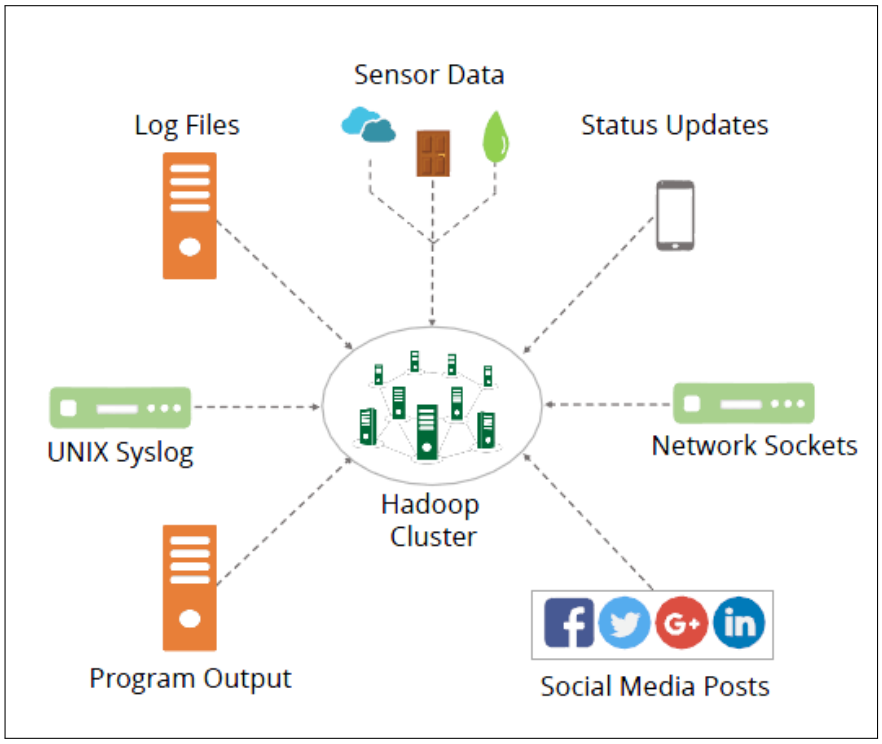


Figure 2.2: The Figure shows different data sources supported by Flume. Flume moves data through memory or disk to finally store into a HDFS based storage location.

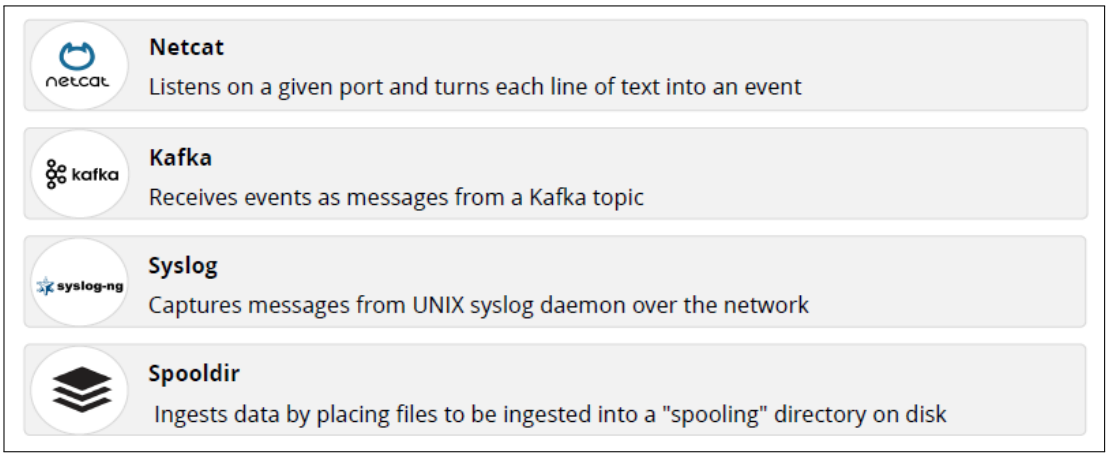


Figure 2.3: The Figure shows different data sources supported by Flume. Flume moves data through memory or disk to finally store into a HDFS based storage location.

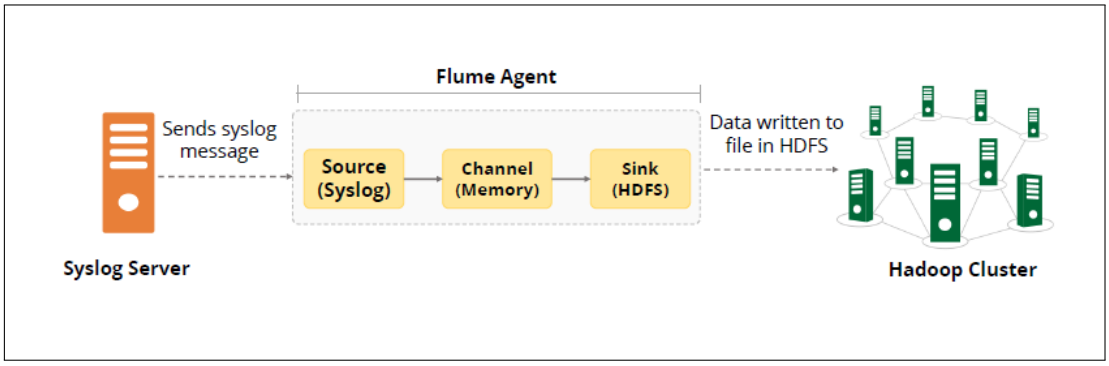


Figure 2.4: End-to-end data flow using Flume. Stream is initially collected from server logs, data transits through memory channel and finally stored into HDFS in a multi-node Hadoop cluster.

developed in LinkedIn and subsequently open-sourced in early 2011. Components of the Kafka cluster include the following—(i) **Broker** is the actual Kafka process. A Kafka cluster can have one or more brokers. (ii) **Topic** are feed names to which the producers publish messages. It can have partitions to increase the degree of parallelism. (iii) **Producer** is an application that publishes data to a topic and in a particular partition within a topic. (iv) **Consumer** is an application that subscribes to a given topic and consumes the feed of published messages. (v) **Zookeeper** is the coordination interface between the Kafka broker and consumers. For each Topic, the Kafka cluster maintains a *partitioned* log. Each *partition* is an ordered, immutable sequence of messages that is continually appended to a commit log. The messages in a partition are each assigned a sequential id number, called an offset, which uniquely identifies each message within the partition. The Kafka cluster retains all published messages whether or not they have been consumed for a configurable period of time or size. Kafka uses partition replication to ensure resilience, as shown in Figure 2.5. Different *consumer groups* and *offset management* in Kafka ensure fault-tolerant distributed computing. If multiple consumers who are part of the same consumer group are hooked to the same Topic, then a message is delivered to only one Consumer within the group. If a message needs to be consumed by multiple consumers, then the consumers have to be part of different consumer groups. Since multiple consumers can consume the same message in a consumer group, the Consumer's responsibility is to maintain a log of what has been consumed so far. The Consumer maintains an

offset which is a position in the log till which messages have been consumed. In earlier versions of Kafka, the offsets were maintained with Zookeeper. From V0.8.1 onwards, the offsets are maintained in a separate topic created by Kafka.

A Kafka cluster can be configured in the following topologies: single node single broker cluster, a single node multiple brokers cluster, and multiple nodes multiple brokers cluster.

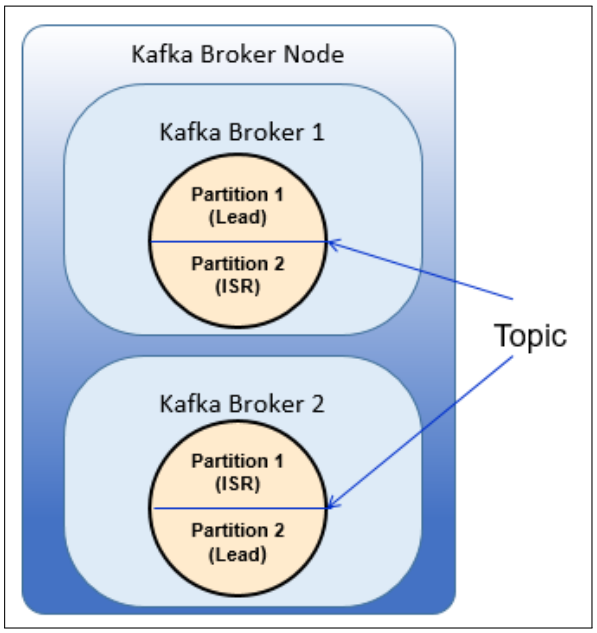


Figure 2.5: Kafka *topic* is partitioned into a *leader* (main copy) of the data, and an *In Sync Replica (ISR)*. *Leader* and *ISR* are hosted into different physical nodes in a cluster where a Kafka broker is running. Assume, that the partition size is 100 mb, then for 200 mb of data load, each of the *lead partition* will store 100 mb data. Replica of *partition 1* in *broker 1* is stored into the *broker 2* as an *ISR*. Similarly, replica of *partition 2* in *broker 2* is stored into the broker 1 as an *ISR*

Apache Kafka witnessed a wide industry adaptation empowered with real-time distributed message queue for the stream processing frameworks [70]. Ingesting a large high-velocity volume of data requires fast, fault-tolerant distributed pipelines. As a distributed client-server-oriented publisher-subscriber messaging system, Apache Kafka replaced many traditional message queue systems like Rabbit MQ, IBM MQ because of its higher throughput, reliability, and replication capability [115].

Adobe Analytics is an industry-leading tool allowing stream data integration. Raw clickstream is accumulated from mobile apps and websites using web service APIs [120] [2]. Adobe Analytics analyses raw clickstream after data is stored and pre-processed in the Adobe warehouse. Adobe Analytics, however, lacks the real-time feature and works on the batch manner for a recurrent hourly or daily schedule.

Several open-source and enterprise stream processing solutions have emerged in the recent past, like Spark Streaming, Storm, Flink, Beam, which is widely explored in web traffic and clickstream analytics [96] [82] [125].

Apache Storm is an industry-leading distributed client-server stream processing framework, written primarily in the Clojure programming language, first developed at the BackType company [121, 232]. The project was then open-sourced to Apache after being acquired by Twitter. An optimisation technique for Storm's default scheduler to perform fair load balancing, especially in heterogeneous Storm clusters setup, was proposed in [77, 278] to overcome the limitations of the default round-robin scheduling, which attempts to overcome the disparity between resource requirements and availability that can lead to poor performance and resource wastage. *Stream groupings* in Storm decides workload allocation among different bolts. There are *eight* stream groupings built into Storm. The *shuffle grouping* is the default scheduler in Storm. For the production deployments, an optimized *stream groupings* is recommended, replacing the default scheduler [237]. To support design justifications in selecting specific data mining approaches and visual analytics in clickstream analysis using multi-levels granularity, in a recent work [157], researchers have proposed visualisation strategies for the sequential forms, raw sequences, and design collaboration methods.

Existing research on clickstream data is not based on the public cloud deployments and can not take advantage of building on top of the state of the art cloud offerings. The proposed methods in Chapter 3, in comparison, are hosted on the Apache Storm cluster in Microsoft Azure HDInsight Cloud environment [60] [44]. As a result of cloud capabilities, the proposed model is robust in failover with no impact on cluster downtime when a worker node is down. The model can build

a streaming pipeline with a host of Azure Services and configurations optimised for a production-ready deployment.

Several application areas were investigated for Storm at the implementation level, like the multi-sensor data fusion principle and a real-time network intrusion detection system [164]. Using UML profile of Storm applications, performance analysis, transformation patterns, and health monitoring provides important benchmarking results [286].

A case study for clickstream analysis through Storm and Microsoft HDInsight cluster presented in Chapter 3 was not introduced before.

The application areas for data stream ingestion include *e-commerce sites*, collecting users' *click to views* data or *clickstreams*. E-commerce sites gather clickstreams with different intent, and the underlying frameworks differ on data volume and tools they use. Clickstream & behavioural analysis with context awareness for e-commerce applications were subsequently studied by researchers [45]. Analysis of sequential pattern mining algorithm on web clickstream data [126] is the most common case study for clickstream data mining. Many applications of clickstream data emerged in recent years. Those include recommender systems [204] [185] [147] and predictive user modeling for forecasting on potential customers and buying suggestions [240]. With respect to the healthcare domain, monitoring physiological stream data such as continuous Electrocardiogram (ECG), Heart Rate (HR) data of patients undergoing anaesthetic was explored for developing a warning system during operations [296]. The physiological stream data management system helps clinicians to track real-time trend patterns and generate alerts.

2.2 NoSQL as the Data Storage

Experiments presented in the thesis required flexible data models and need to ensure datastore scaling up in case of failover through replication. Therefore, a review of *two NoSQL databases (HBase and Cassandra)*, concerning the type of setup used in this research is given here.

2.2.1 HBase Architecture

HBase consists of *three* major components: *Client, HBaseMasters, and Region Servers* [85, 258]. Region Servers can be added or removed based on requirements. Clients connect to the *Region Servers* directly for accessing data. Region assignment and Data Definition Language (DDL) operation (create, delete, updates) operations are handled by *HBase Master* demon. *Region Servers* are the slave component which are installed along with Hadoop *datanode*. Typically per cluster node, one is configured as *Region Server*. HBase Tables are sharded horizontally by row key range into *Region*. A *region* contains all rows in the table between the region's start key and end key. *Regions* are assigned to the data nodes in the cluster, called *Region Servers* that serve data reads and writes. A region server can serve about 1,000 regions. Data tables are split to become regions. *Regions* store a range of key-value pairs, and each *Region Server* manages a configurable number of regions. The default region size is *256 MB*. *HBase Master* is responsible for *Region* assignment, DDL operations (create, update, delete tales, etc) operations (Figure 2.62.7). *Zoopkeeper* is a part of HDFS, maintains a live cluster state used for coordination and monitoring between clients and server. It maintains the state of the servers (alive and available), and provides server failure notification (Figure 2.8.

HBase contains a *meta table* that keeps a list of all regions in the system. The *meta table* is key-value formatted (RowKey and RegionServer) that holds the location of regions on the cluster. Zookeeper stores the location to this meta table. The client gets a reference of Region Server hosting the *meta table* in Zookeeper. The client queries the *meta table* to retrieve the region server corresponding to the row key it needs to access. The client *caches* the data address along with the *meta table* location for reading data (Figure 2.9).

2.2.2 Cassandra Architecture

Cassandra datastore is a *column family* oriented NoSQL database [58, 105]. A *column family* can have any number of columns. Columns in RDBMS define the type of data that can be stored. Column families have no such restrictions, and they can contain any number of columns and any

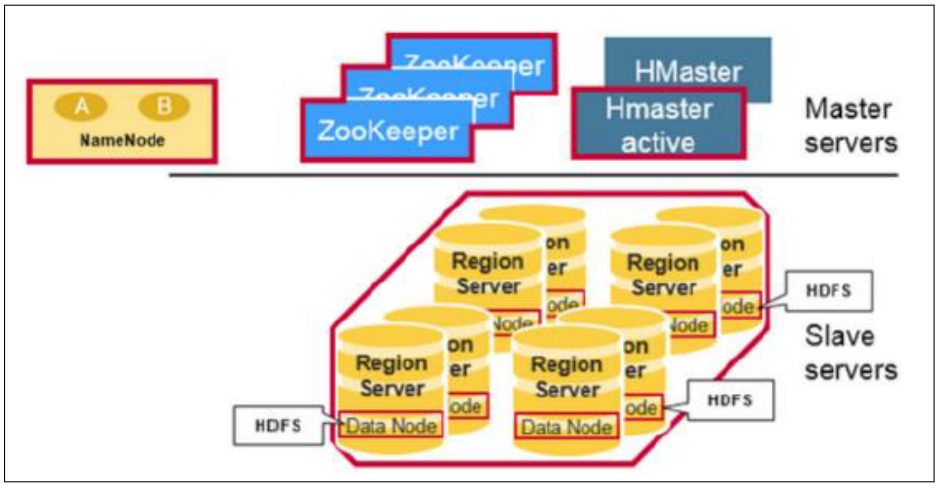


Figure 2.6: HBase architecture high level view.

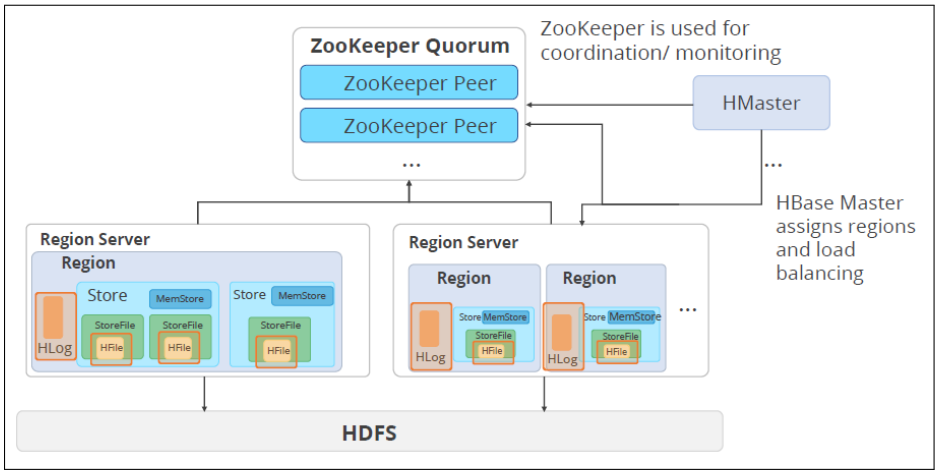


Figure 2.7: HBase architecture with a detailed view of the components of Region Servers.

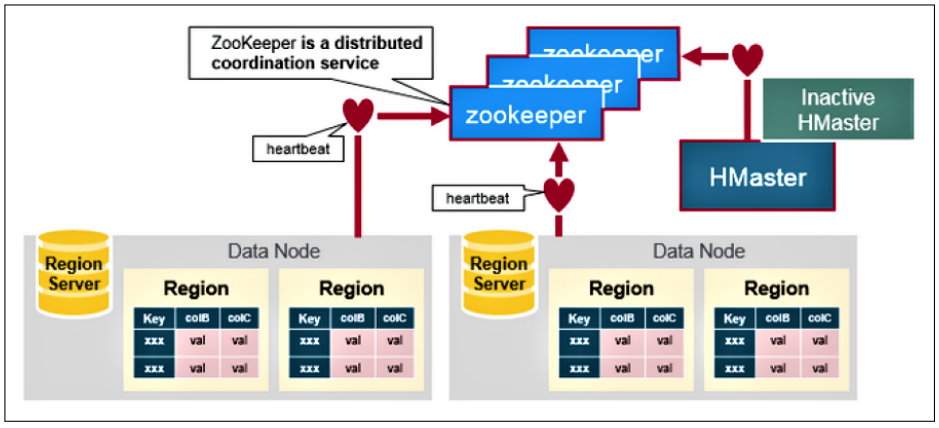


Figure 2.8: Zookeeper architecture.

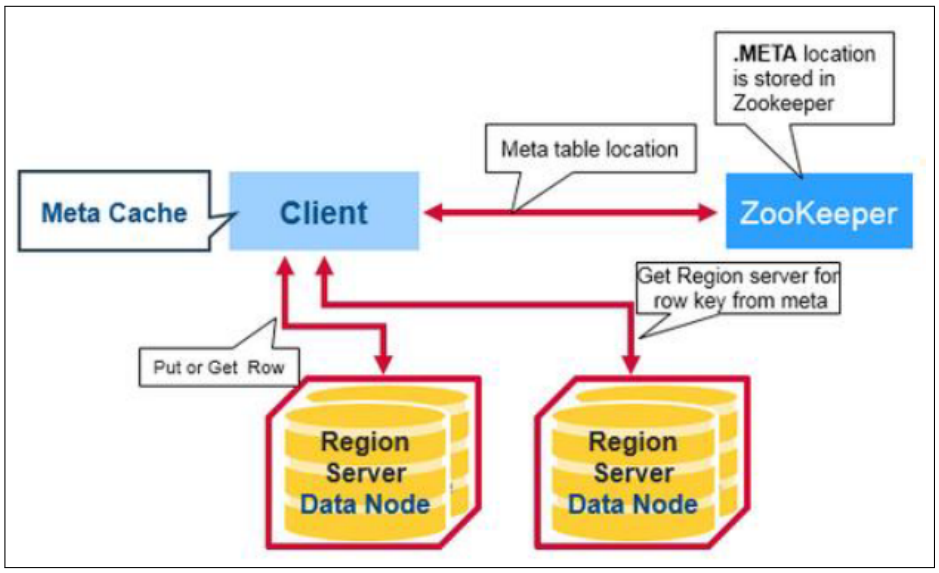


Figure 2.9: Zookeeper meta table.

type of data. Each row of a column-oriented database stores data in only those columns for which it has valid values. Unlike RDBMS, *null values* are not stored at all [105, 174].

The architecture of Cassandra horizontally scales and performs with continuous availability. It has a *master-less ring* distributed architecture and easy to set up and maintain. Cassandra's built-for-scale architecture is capable of handling large amounts of data and thousands of concurrent users/operations per second across multiple data centres easily.

Gossip Protocol in Cassandra is a peer-to-peer communication protocol in which nodes can choose among themselves to exchange their state information. Each node has *some* data associated with it and periodically gossips this data with another node. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster. Cassandra distributes the data across the entire cluster. Data is stored on nodes in partitions, each identified by a partition Key, and distributed across the cluster by the value of a Token. Cassandra uses *partition* to distribute data across the cluster. *Partition* is *a hash function* located on each node which performs hashing operation on partition. *Token* is an integer value generated by a hashing algorithm, identifying a partition's location within a cluster. *Murmur3Partitioner* is the default partitioner in Cassandra. A *Snitch* determines which datacenters and racks nodes belong to. *Snitch* inform Cassandra about the network topology and allows Cassandra to distribute replicas. *Snitch* determines relative host proximity for each node in a cluster, used to determine which nodes to read and write for the fastest operation.

Cassandra represents the data managed by a cluster as a ring. Each node in the ring is assigned with one or more data ranges as described by a Token, which determines its position in the ring. A Token is a 64-bit integer used to identify each partition, providing a possible range for Tokens from -2^{63} to $2^{63}-1$. A node claims ownership of the range of values less than or equal to each Token and greater than the Token of the previous node. Figure 2.10 shows a notional ring layout including the nodes in a single data centre. The particular arrangement is structured such that consecutive Token ranges are spread across nodes in different racks.

Virtual nodes Token allocation is as follows: The Token range from -8 to 7 is distributed

among four nodes as shown in Figure 2.11. Node A has allocated Tokens with values greater than or equal to -8 and less than -4. Node B has Tokens for a range greater than or equal to -4 and less than 0. Node C has Tokens with a value greater than or equal to 0 and less than 4, and node D has a range greater than equal to 4 and less than 8. Within a cluster, virtual nodes are randomly selected and non-contiguous. The placement of a row is determined by the hash of the partition key within many smaller partition ranges belongs to each node (Figure 2.12.

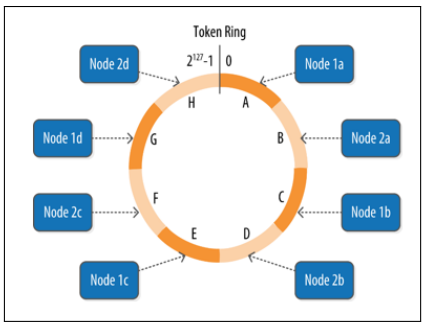


Figure 2.10: Cassandra notional ring layout including the nodes in a single data center.

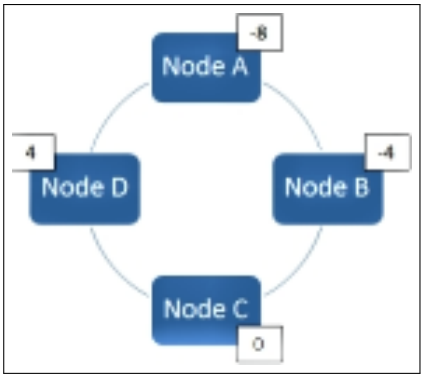


Figure 2.11: The Token range from -8 to 7 is distributed among four node

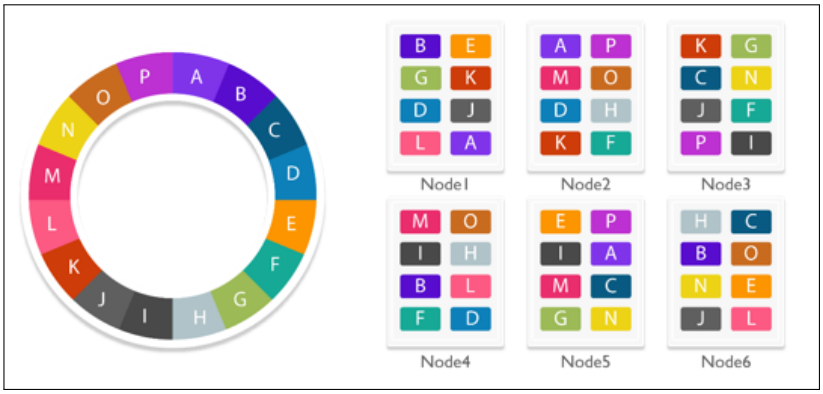


Figure 2.12: Reading data through a coordinator node.

Reading data through coordinator node. If a client connects to a node that *does not* have the data it is trying to read, the node will act as a *coordinator* node. A client requests data from

a coordinator node sending a request to all replica nodes responsible for owning the data. The coordinator node compares the data and responds to the client with the most recent data returned by the replicas as shown in Figure 2.13.

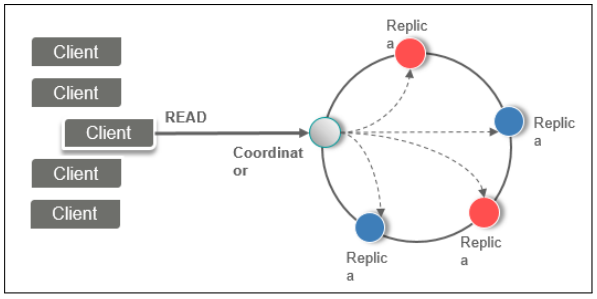


Figure 2.13: Reading data through a coordinator node.

Hinted Handoff is a mechanism to ensure availability, fault-tolerance, and graceful degradation in Cassandra. It consists of the following information: (i) a target node location, which is down, (ii) partition, which requires a replay, (iii) data needs to be written. If the replica node where the write belongs fails, the coordinator will create a *hint*.

Because of *Gossip* protocol, the coordinator node receives the hint and has the knowledge about the unavailable node coming back online again. Coordinator stores the hint in its *system.hints* table. *Five* sequence of stages for the *hinted handoff* process as shown in the Figure 2.14 :

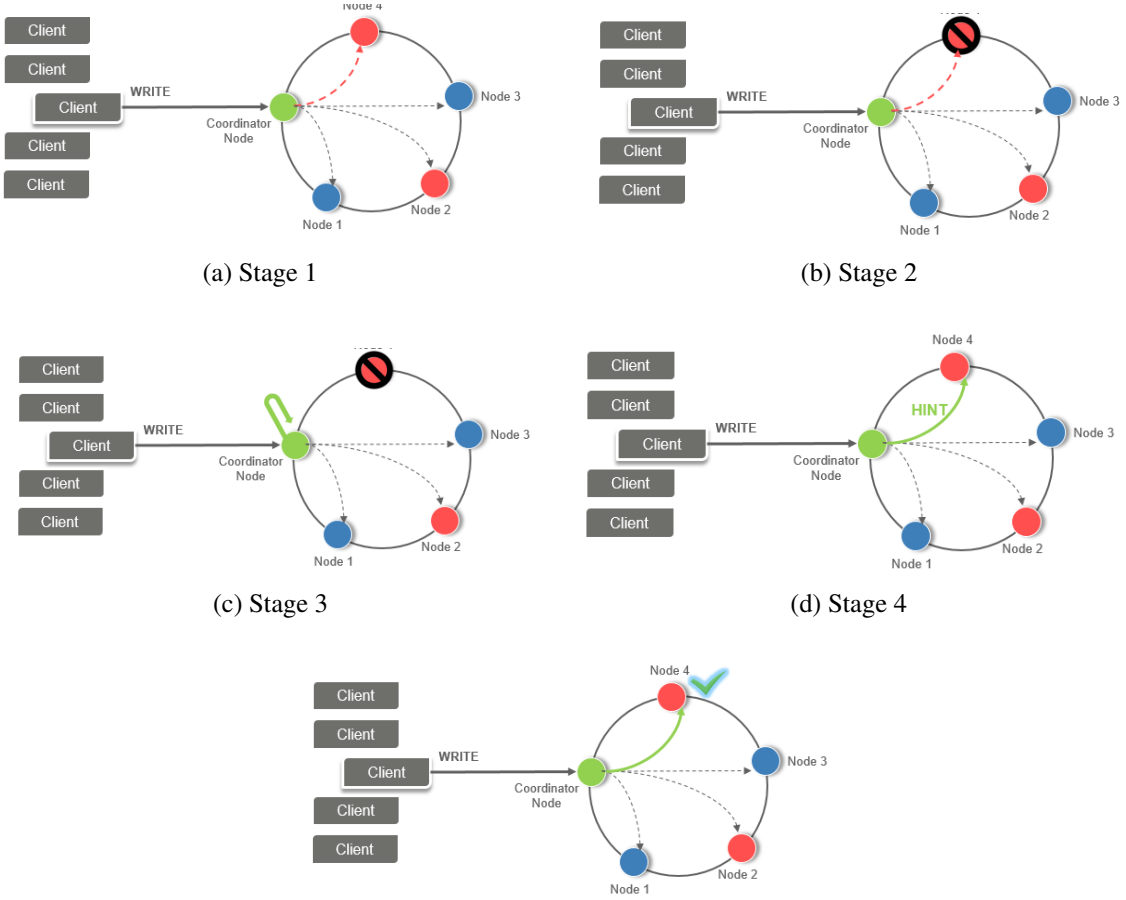
Stage 1: At this step, data needs to be written on *node 4*, for that, write. Request has been given to *Coordinator Node*.

Stage 2: If *node 4* becomes unavailable due to a system or network failure, write operation *can not* be performed at the node.

Stage 3: *Coordinator node* temporarily stores the write to itself on behalf of the node.

Stage 4: Once *Node 4* is back into operation, the *Coordinator Node* forwards the request to node 4 as a *hint*.

Stage 5: The write operation is executed on node 4. Figure 2.14 illustrates each stages.



(a) Stage 1

(b) Stage 2

(c) Stage 3

(d) Stage 4

(e) Stage 5

Figure 2.14: Different stages of *hinted handoff* through a coordinator node.

2.3 Big Data Lambda Architecture

The thesis presents a collaborative *ensemble* environment that makes it easier to tap into the power of batch and real-time analytics. The proposed model is developed on top of the standard Lambda Architecture (LA) in big data analytics. LA is a general-purpose, fault-tolerant big data *hybrid* processing model first coined by Nathan Marz [167] from Twitter. Since then, LA has witnessed wide industry adoption, especially in the e-commerce domain in companies like Amazon and Best Buy, to leverage both batch and stream processing paradigm for big data analytics.

LA [102] combines both batch and stream processing approaches. As depicted in Figure 2.15, the full dataset ingested to the system is moved to both batch and stream layers for processing. The stream layer serves only low-latency queries. Data gets merged for a type of data mining task that requires historical data.

The architecture consists of three layers:

1. **Batch layer**

   This is a high-throughput/high-latency option. Processing duration varies from a few minutes to hours. Data is ingested in large batches at a certain schedule; reports are generated, users check at the same reports until the next data load occurs. However, the batch layer can *not* have access to real-time data. The speed layer resolves the issue. The Frameworks and solutions like—Hadoop MapReduce, Spark core, Spark SQL, GraphX, and MLLib are the widely adapted big data tools using batch mode [161,281]. Batch schedulers include Apache Oozie [118], Spring Batch [65], and Unix Corn which invokes the processing at a periodic interval.

2. **Speed Layer**

   This is a continuous non-blocking option for low latency messaging and event processes responding to users' requests in real-time or near-real-time. Most operations on streams are windowed operations operating on slices of time like moving averages for the stock process

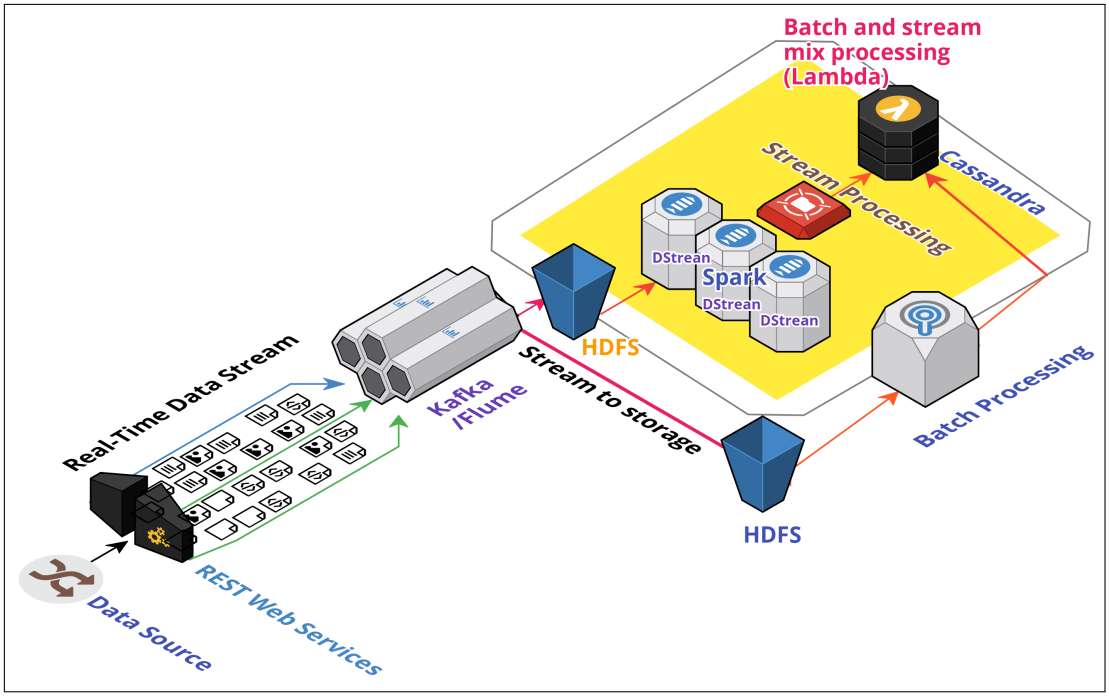


Figure 2.15: Lambda Architecture combines low latency real-time frameworks with high throughput Hadoop batch framework over massively distributed setup. Observe, while data is processed in real-time as Spark DStreams, simultaneous batch processing occurs through the stored data from HDFS. Cassandra stores the combined view from batch and stream.

every hour, top products sold this week, etc. The speed layer processes the most recent data but in a limited time frame. To retrieve a holistic view of the data, both real-time and batch views must be aggregated through another batch process. Popular choices for stream processing tools include Apache Kafka, Apache Flume, Apache Storm, Spark Streaming, Apache Flink, Amazon Kinesis, etc. [37] [254].

3. **Serving Layer**

   The serving layer consolidates batch and real-time views into one and retrieves results in real-time for on-demand queries over the entire dataset. It provides low-latency access to the full dataset.

Features of the Lambda Architecture are described as follows:

1. Incoming data is transmitted to both the real-time and the batch layer *simultaneously*.
2. The batch module consists of an append-only, immutable master dataset.
3. The speed layer removes the latency issues of the batch layer and has access to the real-time data only.
4. Two layers consist of separate configuration files and their processing logic but the data schema on which it works is generally the same as long as they process on the same data.
5. The serving layer can index the data for faster retrievals.
6. The architecture bears the limitation of data redundancy in two layers and often deal with the duplicate code base in both the layers. Changes in one layer require updates in another layer.

A number of architectures were introduced for LA involving applications dealing with a high velocity of data ingestion. Scalable stream processing was started as a new category of open source project by Twitter's Nathan Marz [275], who also designed the generic model for LA. In their work, Yamato *et al.* [283] illustrated a model for real-time predictive maintenance on an IoT deployment

for the company NTT DOCOMO. The model analyses the sensor stream data for anomaly at the speed layer, which requires prompt actions. In the batch layer, raw data are sent to the cloud and stored in the DB using low-cost methods such as night transfers. A maintenance application predicts failure to analyse data in detail, which does not need prompt actions. In the area of hybrid active learning, Kim *et al.* [130] extended the ideas like online learning algorithms and batch processing to propose a model to integrate pool-based and stream-based sampling strategies for active learning, addressing the scenarios where concept drift is prevalent, and labelling is asynchronous.

Few notable domain application areas have emerged through these for LA like telecom, e-commerce, social networking, and manufacturing [183]. Application areas of LA in the field of E-Commerce are studied in works [184] [186] which is studied by the proposed model in the thesis.

Lee *et al.* [143] implemented the LA model for a restaurant recommender system. Their work builds upon several open-source software like Apache Mesos, Kafka, and Spark. At the speed layer, the pipeline processes the incoming stream data to compute users' ratings. The batch layer is designed to process the large data offline and execute complex machine learning algorithms using Spark.

Apache Storm and Spark are two prominent Stream processing platforms for big data. Batyunk *et al.* [46] show an LA implementation for processing streaming data from social networks using Apache Storm to perform the task to build up a predictive model of trends based on data stream from GitHub and Twitter. Hanif *et al.* [97] proposed an adaptive watermarking and dynamic buffering timeout mechanism for Apache Flink, which is designed to increase the overall throughput by making the watermarks of the system adaptive according to the input workload.

However, the Lambda Architecture leads to duplicate data processing in two layers with batch and stream needs to maintain identical codebase in both the layers that lead to redundancy, additional system resource utilisation and undesirable complexity. The framework also lacks collaboration between two layers and does not contain a decision-making component. This research introduces a unique multi-agent collaboration between two layers that leads to developing an *Incremental Lifelong Learning* model. Also the proposed model introduces a decision making com-

ponent between *ingestion* and *processing* layers.

2.3.1 Kappa Architecture

An alternative variant to Lambda Architecture, Kappa Architecture [153, 300] was proposed. The disadvantage of maintaining two processing layers is addressed through *kappa Architecture*. The main distinction in the Kappa Architecture is that it contains multiple stream processing engines. Each stream processing unit is associated with a separate configuration file and manages a separate codebase for each engine, as shown in Figure 2.16. In the case of re-processing due to updates in the data, a separate stream processing job is initiated in parallel with the regular stream job. This way, the entire master dataset containing the history data can be processed ad-hoc through the stream engine itself without the need for a separate batch layer.

The advantage of the Kappa Architecture is, it can work without a batch layer. However, the cluster size and the hardware requirements remain the same as LA when a large volume of historical data is processed in parallel along with the stream data using multiple stream engines. Also, Kappa Architecture lacks processing capabilities for a large set of the static historical data pool, which only suits the limited type of case studies.

The proposed framework in Chapters 5 6 builds on top of the generic architecture provided by Lambda Architecture with additional layers of a decision-making component, a knowledge base, and a knowledge miner. The framework also considers real-time and batches components as autonomous, self-organising, collaborative Multi-agent systems; see Chapter 5 for details. There is a number of variants of Lambda Architecture, and not all the implementations duplicate the same task in both layers. For instance, the recommender system, a case study presented later in the thesis, has the speed layer that computes *browsing history* and the batch layer simultaneously builds a *list of recommended items*. Therefore, the two layers are autonomous and produce independent views of the data. A knowledge miner component as a serving layer *blends* both batch and speed modules, with more recent items in the browsing history getting higher weight while producing the final list of recommendations.

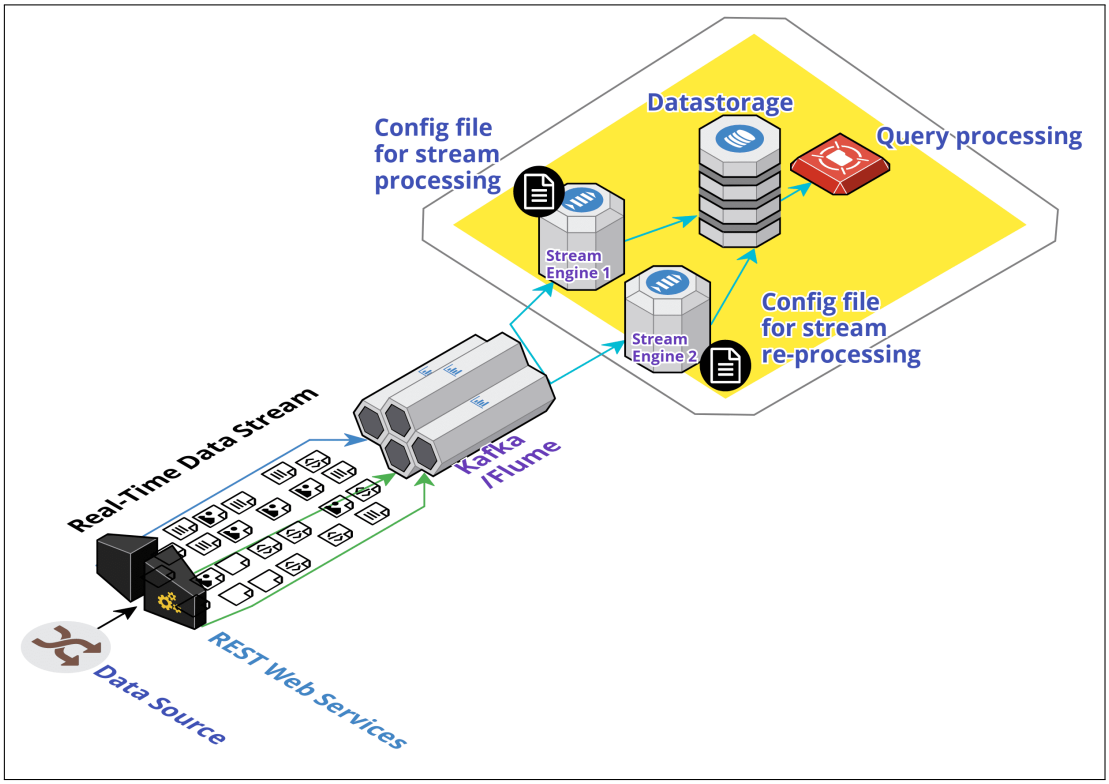


Figure 2.16: Kappa Architecture consisting of only a stream layer. The architecture consists of multiple stream processing engines, each associated with a separate configuration.

Chapter 5, 6 use the streaming K-means library distributed by the Apache Spark project. The idea for a streaming version of K-Means clustering is to split the stream into mini-batch and to integrate the learning of the previous window into the next ones [222]. Hence, in the case of online K-means, the model is reconstructed with each fleeting window as a mix between cluster centres computed from the earlier batches and the following one. The algorithm initiates with assigning data points to their closest cluster. For each new repetition, with the latest data arrival, the algorithm calculates new cluster midpoints, then change each cluster by using the following rule:

$$a_{t+1} = \frac{a_t p_t \mu + b_t q_t}{p_t \mu + q_t} \tag{2.1}$$

$$p_{t+1} = p_t + q_t \tag{2.2}$$

Where p_t is the old data samples and a_t is the old cluster midpoint. q_t is the new data samples and b_t is the new cluster midpoint. μ is the decay factor. If μ=1 all data is used since the inception of the batch, with μ=0, only the most recent data sample will be considered.

2.4 Collaborative Multi-agent Framework for Big Data

The Big Data community had attempted to solve the problem of lack of collaboration, high throughput and low latency in the stream and batch module through LA and Kappa. However, the architecture fails to deliver the same level of efficiency in *disparate* data formats best satisfying a set of criteria. Therefore, a proposed decision-making component through a graph-based model is introduced to increase overall efficiency in the case of *data variety* problem.

The work presented in Chapter 4 brings about many novel concepts in the *Multi-criteria Decision Making Problem (MCDM) problem* with the introduction of a Fuzzy graph-based implementation and techniques to compute the collective influence of multiple coexisting criteria. This section explores the related work in the area of multi-criteria decision-making problems.

A multi-agent system consists of multiple collaborating software components or *agents*. Soft-

ware agents are characterised by two basic capabilities: autonomy and flexibility, which make multi-agent technology well suited for implementing distributed, big data and real-time applications [43]. In a strategic interaction of a society of n agents, each agent's influence depends on its action as well as the action of its neighbours. Methods are explored to analyse the collective influence for such a society of interconnected agents [83]. The model is extended to propose a graph-based Multi-Criteria Decision Making (MCDM) framework [290]. MCDM consists of a set of alternatives that performs a plan as defined by a set of criteria. For an alternative, the criteria values are aggregated to produce an overall value. All the alternatives are then compared to choose the best one by the agents themselves. The model can be implemented through a software engineering improvement project where the alternatives are a set of software projects to choose from. Criteria are a set of factors like —differentiation, productivity, and management concerning different software applications (more detail in Chapter 4). A Multi-agent Coalitions Decision Making (MACDM) scheme [289] describes two competing groups that aim to improve their own benefit with an explicit aim to *outsmart* the others and maximise their own gain. In MCDM, a number of stakeholders are involved in choosing a single solution from a set of available solution options. It is common for the stakeholders to form a coalition during negotiation in order to increase their individual welfare [267]. The framework is implemented through directed sub-graph and integer programming models. While the model is developed for the best strategy of a specific camp to maximise the advantage over the opponent a group of agents, the model, however, suffers in a dynamic MACDM scenario when decision factors, agents' strategies, and damages frequently change with a real-time dataset.

A fuzzy graph-based model represents multiple agent interactions in the thesis as fuzzy graphs faithfully capture real-world uncertainties. Mordeson first studied Fuzzy graph functions [179] in 1994, and later Sunitha [241] explored compliments of fuzzy graphs and related properties. The strength of a path in a fuzzy graph is classified into categories like α, β and δ (stronger to weaker paths) to explain the structure of a fuzzy graph. Neutrosophic sets (NSs) proposed by Smarandache [234] is a generalisation of fuzzy sets and intuitionistic fuzzy sets which can

be used to analyse incomplete, indeterminate and inconsistent information that exist in the real world. A framework for multi-criteria group decision-making (MCGDM) problem is proposed using NSs and dependent functions such as interval numbers, interval dependent function where the distribution function used to describe inherent distribution information of an NSs [282]. A transformation operator is constructed to convert SNNs into interval numbers, and the interval dependent function for SNNs is built, which effectively decreases original information loss. Based on that, the dependent degree of each alternative solution is provided along with uncertainty and stability outcomes for each alternative.

Chapter 4 extend the fuzzy graph fundamentals like *fuzzy vertex* and *fuzzy edge*, *path*, *diameter* and *strength*. The strength of the fuzzy graph represents a fairly consistent value compared to an individual graph edge. Hence the proposed model in Chapter 4 uses *strength* of the fuzzy graph for computing mutual influences between the criteria.

2.5 Lifelong Incremental Learning

The thesis introduces Lifelong Machine Learning through incremental updating of stream data over batch using LA. Lifelong Machine Learning can continually accumulate knowledge during the learning phase as a never-ending learning process. The knowledge extraction and reuse abilities enable lifelong machine learning to solve related *similar* sub-tasks in a problem domain. The traditional approaches like Naïve Bayes and few neural network-based approaches only aim to achieve the best performance upon a single task [192, 219]. The requirement for labelled data for training can also be significantly decreased with knowledge reuse. Unlike previous approaches, the lifelong machine learning presented in Chapter 5 focuses on how to accumulate knowledge during learning and leverage them for further tasks on a continuous basis. Proposed methods suggest that the overall objective of lifelong learning is to use less labelled data and trades off with the computational cost to achieve the performance even better than supervised learning.

Over the past 30 years, machine learning has achieved significant development. However, it is still in an era of *Weak AI* rather than *Strong AI*. Current machine learning algorithms only know

how to solve a specific problem without the knowledge of reusing past learning to solve a group of *related* problems. Therefore, lifelong machine learning (or simply lifelong learning) [249] was introduced to solve the difficulties of knowledge accumulation and reuse with an infinite sequence of related tasks. For a group of related problems, an integrated model with knowledge reusing could decrease the cost of computing, such as *sample annotation* in sentiment classification. In the *sentiment classification*, to predict the sentiment (positive or negative) of a sentence or a document, traditional approaches need to train an independent model on each domain to obtain the best performance. Each domain needs to collect labelled data for supervised learning. This approach is considered as *weak AI*.

To achieve the goal of *strong AI*, we need to change the learning goal to understand the sentiment polarity in words. This means that the algorithm should know how each word influences the overall sentiment of a document in different tasks. If we can achieve this learning goal, the algorithms can solve new tasks without further training. Zhiyuan Chen [64] proposed an approach towards lifelong learning, but a *supervised learning* was still required. Guangyi Lv [159] extends the work of [64] with a neural network-based approach. However, supervised learning is still necessary under their setting, and a huge volume of labelled data is necessary. Hence, the proposed method in Chapter 5 aims to decrease the usage of labelled data while maintaining performance.

The work presented in Chapter 5 addresses the problem using a Lifelong Machine Learning model that is analogous to several machine learning models such as lifelong learning, incremental learning, multi-task learning, transfer learning, and streaming learning. Thrun [247] first studied supervised lifelong learning through the decade of the 90s. The work explored information sharing across multiple collaborating tasks through neural network binary classification. A neural network approach to LML was introduced and subsequently improved by Silver et al. through the years 1996 to 2015 [227] [229] [230] [228] [110] [109]. Cumulative learning is explored in the form of LML, which builds a classifier to classify all the previous and the new classes by updating the old classifier [80]. Ruvolo et al. [215] proposed an LML system based on the multi-task learning developed by Kumar et al. Efficient Lifelong Machine Learning (ELLA) [216] presented by Ruvolo

and Eaton. Compared with the multi-task learning [59], ELLA is much more efficient due to its unique proposition with real-time learning that gradually refines the learning outcome over time to maximise performance across all tasks. Zhiyuan, etc. [64] improved the sentiment classification by involving knowledge. The object function was modified with two penalty terms corresponding with previous tasks. [138]. Here, the learning tasks are autonomous and distributed. In the area of lifelong Unsupervised Learning, Zhiyuan et al. Wang [265] proposed various lifelong modelling techniques to generate topics from a set of historical tasks and use past knowledge to develop better topics. A notable application area like the item recommender system using LML emerged [156]. In the field of lifelong Semi-Supervised Learning, a Never-Ending Language Learner is proposed by Carlson et al. [57], and Mitchell et al. [175]. Using continuous web crawling, a large volume of information is gathered representing entities and relations in this approach. A testing scheme for the LML system is proposed by Lianghao et al. [148], where the incremental learning ability of LML is tested to verify if the system becomes gradually more knowledgeable over time through accumulation, transfer of knowledge in each iteration. Leveraging the previous research in the area of LML, the proposed model uses an Incremental Lifelong Learning (ILML) approach through Spark Streaming which initialises itself with batch offset at the starting.

Chapter 5 leverage Spark Streaming APIs in the proposed MALA architecture for lifelong learning architecture. Spark Streaming provides an abstraction on streaming datasets called *DStreams*, or discretised streams. *DStream* is a sequence of data arriving over time [218] [235]. Internally, each DStream is represented as a sequence of RDDs arriving at each time step (hence the name *discretised*). DStreams can be created from various input sources, such as Flume, Kafka, or HDFS. Once built, they offer two types of operations: transformations, which yield a new *DStream*, and output operations, which write data to an external system. *DStreams* provide many of the same operations available on RDDs and additionally provide new operations related to time, such as sliding windows [212]. Spark MLlib library includes APIs for sliding window-based clustering on streaming data.

Datastream clustering was widely researched and improved over the years. Among the early

works in this area, Guha [89] proposed the *STREAM* algorithm, which produces a constant factor approximation for the k-Median problem in a single pass and using small space. Gupta et al. [92] presented a study for outlier detection approaches and case studies for different forms of temporal data. A time decay function that puts variable weights decreasing over time while updating the model is suggested. The proposed model extends the *STREAM* algorithm with an enhancement that allows further merging of a large static historical data pool with the latest and most updated streaming model.

Various improvement techniques were proposed for dimension reduction in large scale high-dimensional data. Agarwal [34] presented a dimension reduction process for the online learning component where only the limited parameters are learned online, and the remaining item features are learned through an offline batch process. The model is proved through a recommender system implementation for Yahoo! front page and My Yahoo!. *Chapter 5* leverages this approach for the LML system. Apache Spark MLlib implements Random Decision Forest (RDF) for APIs supporting binary and multiclass classification and also for regression models. APIs can handle both numerical and categorical features extending existing MLlib Decision Tree APIs [62]. However, the APIs are computationally infeasible for repeated model re-training with high dimensional data in a streaming setting. To improve the efficiency of the Random Decision Forest algorithm and speed up model updation, Chapter 5 proposes an online dimension reduction and data sampling technique on Spark DStreams, which improves each re-training time by reducing the volume of the training dataset.

Through the period of the last two decades, there has been significant progress in machine learning algorithms and frameworks. However, much less emphasis was put on how these methods and algorithms can be used to train over an extended period of time to incrementally become more knowledgeable through knowledge retainment and transfer. Chapter 5 addressed these research gaps by developing an incremental, transfer learning model through a big data stream-batch mix processing approach. A novel collaborative mixed processing framework called Multi-agent Lambda Architecture(MALA) model is proposed, which uses Hadoop MapReduce at the batch

layer and Apache Storm at the stream layer to develop a *lifelong learning machine*. Also, the current predictive analysis frameworks provide forecasting without much explanation and root cause analysis. The proposed model addressed these research gaps between analytics and more explicit *actionable insights* through reasoning and a truly lifelong learning platform. The framework is effective in high dimensional big data applications through its Apache spark in-memory data processing capability, unique dimension reduction technique, and incremental never-ending learning approach. Also, Chapter 5 intends to fill the research gaps in real-time machine learning for so-called *continuous lifelong learning*, where both training the model, as well as predictions, happen in real-time or near real-time.

As a precursor to the current work in Chapter 5, [188] first introduces the concept of MALA in the context of Lifelong Learning architecture. [190] further improves the MALA architecture through a novel scheme for reasoning and root cause analysis, streaming *hybrid* clustering, dimension reduction techniques, and in-memory processing capabilities.

2.6 Background of Implicit Feedback Based Recommender System

In Chapter 6, an *Ensemble Learning* model is introduced to improve Lifelong Machine Learning results. As proof of the proposed ensemble methods, the case study of a *Recommender System* demonstrates that Lifelong Machine Learning results can be further improved through *ensemble* of multiple parameters like context, time etc.

In real-world applications, it is common to have access to implicit feedback in the form of views, clicks, purchases, likes, shares, comments, etc. [3]. Still, overwhelming numbers of existing researches just focus on users' explicit feedback (ratings). Few works [112] explored developing a personalised recommendation engine without users' explicit ratings, which are completely based on implicit feedback like users' clicks. The majority of the existing approaches for recommender systems focus on recommending the most relevant items to individual users without considering any contextual information such as time, place, etc. For example, in online shopping portals, buying pattern is largely influenced by a user's geographic location, such as colder states vs warmer

states, remote areas vs cities, browsing from desktop vs mobile, etc. Recommender systems deal with two types of entities *users and items*. Nevertheless, existing research does not put them into a *context* when providing recommendations. The *context* is a multifaceted concept that has been studied across various research disciplines including Computer Science, Cognitive Science, Linguistics, Philosophy, Psychology etc. [71,128,201,208,262]. Existing Item-to-user rating-based recommender system suffers from issues like user cold start, item cold start, sparsity, scalability, etc. [210,223,279] In user cold start, ratings can not be predicted for a new item until similar users have rated the same item. In the case of sparsity, very few items are rated by users leading to a sparse item-user matrix, and hence finding similar users is difficult. Scalability issue is present as the dataset grows with time, filling the sparse matrix becomes gradually extremely compute-intensive, and each batch run takes a longer time [210,223,279].

The proposed Implicit Feedback Based Recommender System (IFBRS) overcomes the above-said issues easily by adopting an item-to-item collaborative filtering approach as it does not attempt to find similar users based on ratings. The model does not have access to ratings provided by the users and do not access users' preferences captured during registration with the portal. Implicit feedback is purely computed in terms of the item viewed by users. The proposed ensemble model considers *context information* as a latent factor to provide more meaningful recommendations depicted as follows: Users $\times$ Items $\times$ Context $\rightarrow$ Implicit Feedback (clicks) Based Recommendations (more detail in Chapter 6).

2.7 Analyzing Geospatial Time Series Data

The proposed ensemble model is implemented to analyse data linked to both location and time, consisting of satellite and aerial images. The case study validates the efficiency of the model in terms of a large volume of geospatial-temporal data. In recent years, big data has been a major application area for analysing geospatial, temporal data, i.e. data that is associated with longitude, latitude, and time information. Mining insights from the voluminous and ever-generating stores of geospatial-temporal big data carries a substantial challenge. This area of big data ana-

lytics, so aptly named since its intrinsic link to location and time, comprises satellite and aerial images, global-scale data. Geo-referenced IoT/sensor systems and big data events are ingested on an array of computing platforms. Its colossal size and the intricacies related to extraction and transformation make it challenging to process, especially at a larger scale for time-critical applications [211]. A brief summary is given of some relevant applications for this domain. Koubarakis *et al.* [284] applied geospatial data for *earth observations* through the images from satellites. Resource Description Framework (RDF) represents linked data and SPARQL for query language focusing more on linked open data and marginally addressing big data aspects. Tan *et al.* [209] analyzed Yangtze River flooding through geospatial data. They have come up with an approach of cloud and multi-agent-based tracking and management system and geospatial computation that inherently resources intensive tasks. The model is implemented in a project involving *Yangtze river*, the longest river in Asia that causes periodic floods along the vast swath of Yangtze valley. Liu [202] provided a grid computing environment to enhance processing prowess exponentially by a grid computing framework. Grid computing is evolving as a prominent development area in scientific research, and such models benefit *geo-computational* systems that encompass obtaining data from remote sensors to accomplish data acquisition, transformation, and enabling workflow management. Praveen *et al.* [129] presented a general overview of geospatial analytics through the Big Data MapReduce framework. Their paper also introduced *location sensitive trend finder* as a possible case study. From there, a few new application areas have also emerged, like, spatial divergence on public blogs through GIS techniques. A strong correlation between *distance to an incident* from the places people are living and general public reactions due to it is established [72]. Social dating leading to geospatial insights became a prominent application area for Big Data analytics. The possibility of disaster recovery using the social media data using *Weibo* platform data and *Weibo APIs* is explored [280].

In real-time forecasts, a response is returned almost instantly: at a milliseconds range. Real-time analytics are generally used to empower predictive competencies within a collaborating web, desktop, or mobile application. A query to a machine learning model formed with Apache Spark

or Amazon ML for forecasts in real-time is by means of the low-latency prediction APIs. The prediction operation accepts a single input in the application payload and responds with the prediction synchronously. This sets it distinct from the batch mode prediction APIs, which asynchronously responds [102].

In the case study implemented in the thesis, real-time taxi demand forecasting was proposed through the Neural Networks based approach [47]. Though the prediction happens in real-time, the model is trained on a large volume on a historical data pool (about four years of NYC taxi trip data) in a classical batch setup. However, the model lacking the inclusion of the recent data samples to update the model continuously. Also, without a cluster like distributed environment, horizontal scalability can a be bottleneck in Big Data scenarios. This work [93] has taken an approach to detect outliers through graph processing and machine learning. The work has made use of historical batch data from the NYC taxi app. The clustering approach, however, is limited to the batch mode of execution. Qian *et al.* [98] presented a dynamic pricing model for taxi trips based on the available historical data. The work considers taxi price as a function of distance travelled and time of the day and modelled in the semi-Markov process [270]. The work studies the *time-of-day fare scheme* which allows price multipliers to adjust tour costs dynamically. Since the work only concentrates on historical data, it is worth extending the model for real-time applications. This involves using real-time taxi trip data and real-time data from Google Maps, which can assist as the input for road overcrowding. Deri et al. [263] provided a study of taxi movement in New York City short trips. With four years of historical data, static depictions of the taxi trips are converted to dynamic ones through Dijkstra's algorithm [261]. These paths are used to extract insights, like usual trip counts, as a representation of taxi flows through the city.

Chapter 6 present a case study on geospatial temporal data. The case study implements the proposed Big Data hybrid data processing Multi-agent Lambda Architecture, which combines low latency real-time frameworks with high throughput Hadoop batch framework over a massively distributed setup. The data for the case study uses New York taxi app data. The work addresses the high latency problem of MapReduce jobs by simultaneously processing at a speed layer to the

requests which require quick turnaround time like real-time taxi demands, anomalous traffic detection, etc. At the same time, the batch layer in parallel provides comprehensive coverage of data by providing insights into taxi usage data. The experimental results illustrate the oracle performance significantly by low latency processing at the speed layer and producing unprecedented analytical insights at the batch layer.

Chapter 7 presents new methods for multimodal Topic Modeling, Clustering, and classification using integration of *three* modalities: language, visuals (image and video) and metadata. An overview of existing researches is given in this section in the field of multimodal integration between language and visual features.

2.8 Multimodal Approach to Integrate Language, Visuals and Metadata

Text-image-video cross-modal correspondence. Modelling heterogeneous modalities in the field of big data and natural language processing have recently attracted significant research interests. A joint embedding between text and image requires the mapping of images to their semantic meaning. Images and video are converted into sentences or *word tags* that describe the visuals best [78, 199, 287]. Convolutional Neural Network (CNN) efficiently generates sentences from the images in a *three* step process. Once image modality is translated into a linguistic modality, the resulting analysis only needs to deals with mono-modal data. In the *language modelling* step, the embedding of each word is stored into the *recurrent layers* in a Recurrent Neural Networks (RNN). In the *image layer*, CNN detects visual features and in the *multimodal layer* image, and language layers are mapped together for a joint representation [165].

The cross-modal integration between multiple layers can take place at an *early* or at a *late* stage. In *early fusion*, individual modalities are converted into numerical feature vectors and are embedded into a dense vector representation. All modalities are combined to generate a joint embedding space by a simple union, element-wise addition, multiplication, or outer product between

vectors. As shown in Figure 2.17, text (X) and visuals (Y) are converted to corresponding vectors h_x and h_y. h_m is the joint vector between them. The *softmax* function normalizes the vector elements where each element value falls in the interval (0,1) and addition of all the elements equals to exactly *one* [149, 292].

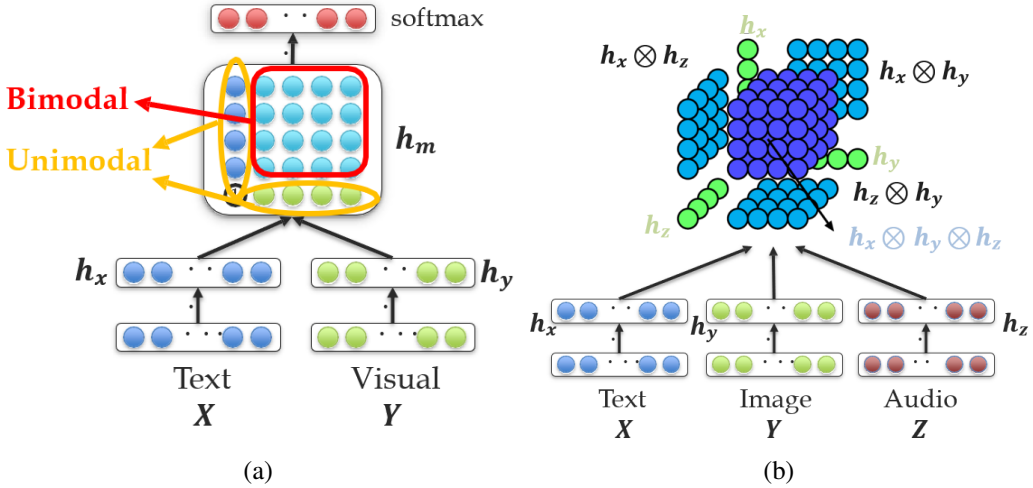


Figure 2.17: **Bimodal and trimodal early fusion.** The symbol $\otimes$ represents the outer product between vectors. Each modality can be interpreted separately after combing them [292]

Early fusion is relatively simple to implement. Since integration takes place at the vector level, it can model low-level interdependencies across features. Although integration of large vectors often leads to a high dimensional vector [1] and integration can be challenging for vectors with non-identical features. In the *late fusion*, modalities are merged after the decision from each modality is obtained from each of the independent data sources. For example, classification through the CNN technique may categorise a video as *mystery* while the associated textual description identifies a category as *entertainment*. Thus, late multimodal integration combines each of the final outcomes to categorise the content as a *mystery movie* using linear weighted sum, or the majority voting method [79]. Early integration, on the contrary, summarises the content before the classification step by adding the feature vectors for all the modalities into a shared representation. Early and late

[1] Dimensionality represents a number of features of the object model as a vector

fusion can use *one-hot vector* or *bag of words* model [81, 124]. In *one-hot vector*, all the words are represented as a single large numerical, sparse vector. The *bag-of-words* model is an orderless set of words within each document. *Hybrid fusion* can exploit strengths of both *early* and *late* fusion (see Figure 2.18, 2.19). In this thesis, we propose a *hybrid integration* technique where text and related metadata are combined at the early stage. The outcome of the early fusion is merged with results from the image analysis; The detailed process is described in Section 7.4.1.

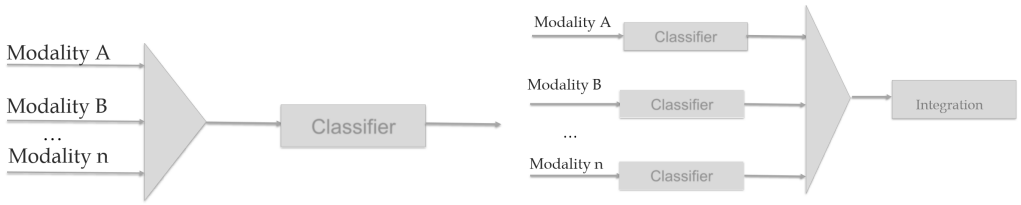


Figure 2.18: **Early and late multimodal fusion.** Early integration combines vectors or raw datasets, while late integration takes place at the decision level.

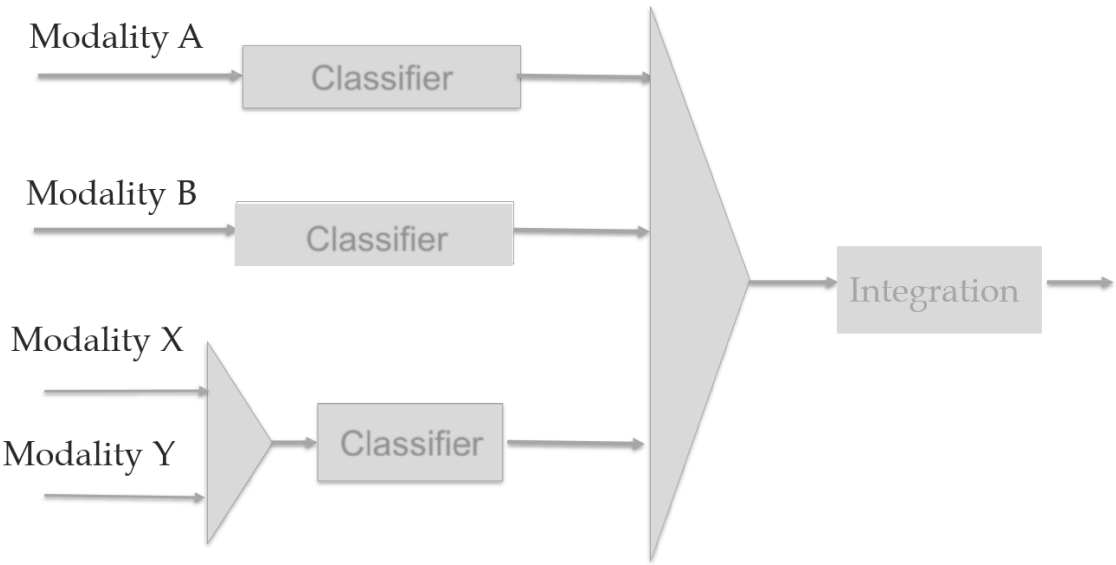


Figure 2.19: **Hybrid multimodal fusion.** The model exploits both early and late fusion models.

Multimodal topic detection. Text-image joint representation can be used to detect *topics* of

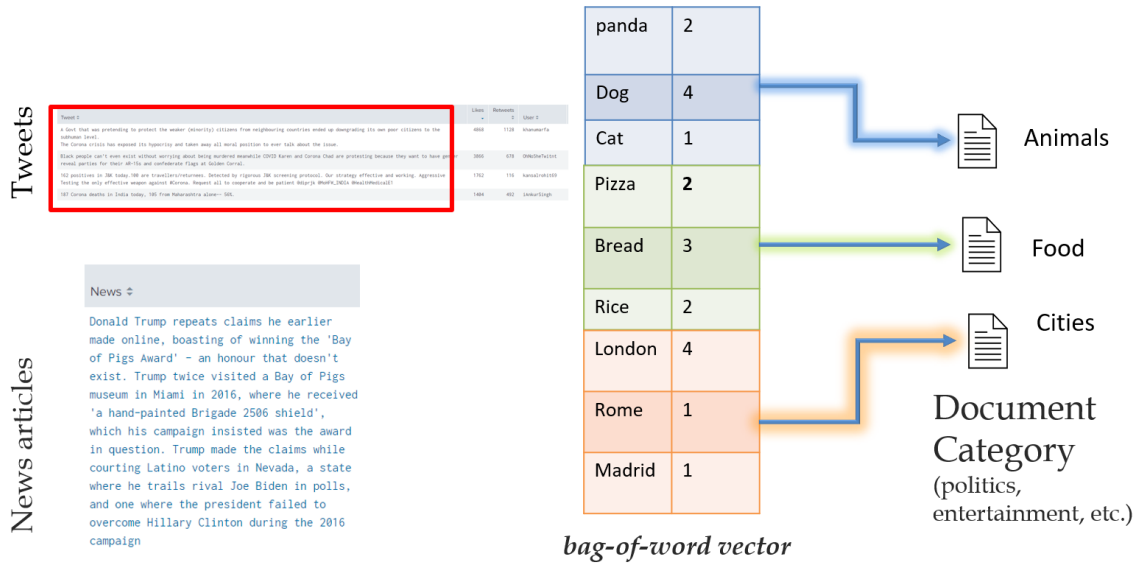


Figure 2.20: **Bag-of-words model.** The model is widely used for document classification

a news article or a social media post. Topic detection eliminates noise by removing unrelated content. Faces in images can be detected using *Viola-Jones face detector*, and other objects in the images are detected using the Local Binary Pattern method [36, 257]. Both faces and objects are converted into visual words using CNN. Named Entity recognition represents *a topic* in each record in the corpus by identifying *"Who", "What", "Where"* components for each text and *"face-Who", "face-What"* for each image for all records in the dataset. A word pair is considered co-occurring if multiple components (e.g. who and where) appear in one single record. Therefore if person *Donald Trump* (W_1) and place *Washington* (W_2) occurs in a record they are considered as co-occurring (W_1 and W_2). Then a joint representation between image and text is computed by the union of image and textual components [151].

Chapter 7 develops a new Multimodal Analytics Platform (MAP) using a case study on big social and news media data. In that context, the state of the art social media analysis platforms are described here.

State of the Art Social Media Analysis Platform

According to *Social Media Monitoring Tools and Services Report* published by Ideya Inc. [28], there are at least 157 social media analysis tools available, the majority being paid subscriptions. However, none of the tools provides an analysis of both news and social media data within the same application. Existing tools analyse the following social media platforms: Twitter, Facebook, Instagram, YouTube, Reddit, Pinterest, Linkedin, and Tumblr. But the same tools cannot *aggregate* the data across the platforms [169, 177, 177]. For example, Twitter and Facebook data cannot be combined together to analyse *aggregated sentiment* of the textual data and compare it with the combined image analysis data. A popular tool like *Sprout Social* [26] analyses Twitter data to evaluate Hashtag performance, track Tweet clicks, and measure Facebook page impact and Linked-in connections evaluate Pinterest post sentiments, etc. However, the majority of the tools lack advanced filtering options. Therefore, they are incapable of complex queries such as the following: *Analyse the Tweets that are related to Donald Trump but not related to Coronavirus and contain both images and videos posted from Washington*. Commercially available tools offer easy to use web-based interfaces that allow users to graphically interact with the system without the need to write software code [41, 73]. Applications, therefore, can easily be used by scientists without any knowledge of data science or programming languages. Nevertheless, in-depth analysis often requires custom functionality that extend beyond the out of the box features that exist within the tools. For this reason, advanced analytics tools should be extendable to incorporate the custom data science logic through Structured Query Language (SQL) or programming languages like Python and Java.

Many existing platforms [90, 207, 244] offer sophisticated image analysis like Reverse Image Lookup. Visual evidence of the number of times an image is used across social media platforms builds a much more powerful metric. Nevertheless, the same frameworks cannot aggregate multiple image searches together. Moreover, images, videos, and textual results are interpreted separately instead of producing a joint multimodal analysis produced by the tool. A comprehensive and *overall* outcome is thus only achieved through a manual process of interpretation from each of the reports separately.

The traditional multimodal analysis is based on data stored in flat-files or Relational Database Management System (RDBMS) as a single standalone desktop [106, 141]. These tools can not offer an effective solution for a collaborative, centralised application where multiple users can simultaneously access the system. Computing processing power requires to scale up incrementally with a fresh inflow of real-time stream data that are added up with existing historical data. Without *indexing* the large volume of textual data, applications fail to achieve a useful interactive reporting experience as each report can take several hours to complete.

In summary, the current applications still fall short of combining multiple modalities in regard to enhancing the objective of multimodal understanding.

2.9 Summary

This chapter presented the current state of the research in terms of data ingestion, in-flight processing, stream batch mix processing, multi-criteria decision making, multimodal integration and lifelong learning. *Kafka* and *Flume* are popular data ingestion tools that ingests data in a distributed and fault-tolerant manner. *Adobe Analytics*, *Apache Storm* and *Spark Streaming* are popular and industry-leading frameworks used for in-flight (between generation and storage) data processing. A brief background of HBase and Cassandra NoSQL data storage is provided with respect to the particular type of setup that is used in this thesis. HBase is client-server architecture divided into *three* components: *Client, HBaseMasters, and Region Servers*. On the contrary, Cassandra uses *master-less ring* distributed architecture. Cassandra *Gossip Protocol* is a peer-to-peer communication protocol, and *Virtual nodes* is the data partitioning mechanism. Lambda Architecture simultaneously processes both batch and stream processing approaches into a single architecture. Kappa Architecture consists of multiple stream processing engines to process both stream and batch data. Multimodal method to integrate language, visuals and metadata uses early (between vectors), late (between decisions), or hybrid approaches to join disparate modalities. The multimodal topic detection process involves converting images to textual annotations, parts-of-speech and named entity recognition of text and associated visual descriptions. Lifelong machine learning

accumulates knowledge over the period and reuses the learning on the infinite sequence of related tasks. In the area of Multi-criteria decision-making methods, the strategic interaction of a society of autonomous agents demonstrates the collective influence of interconnected agents. Finally, a relevant background of case studies is presented that are used in this thesis to validate the proposed methods, such as recommender system, geospatial and social media analysis.

Chapter 3

Getting Data In: Ingesting Data through Distributed Big Data Frameworks

3.1 Introduction

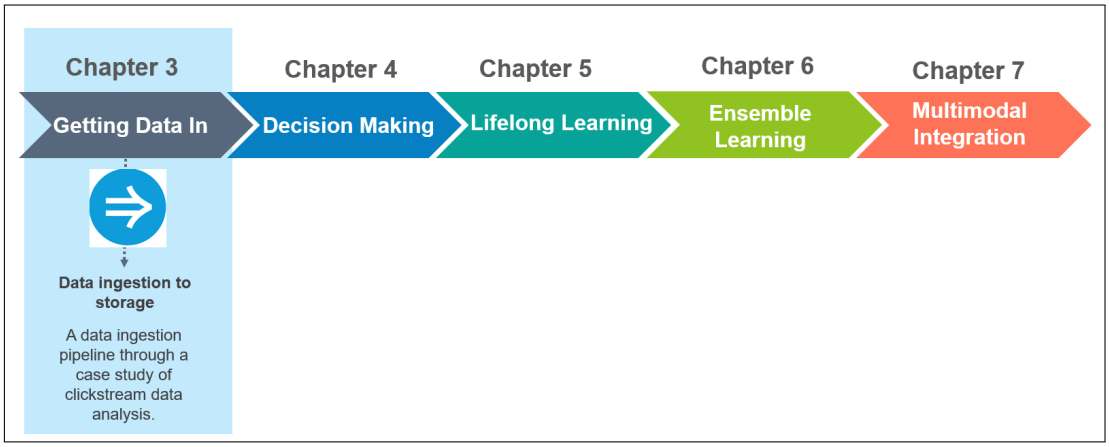


Figure 3.1: An outline of Chapter 3 and end-to-end data flow pipeline in different chapters.

This chapter presents a high velocity, fault-tolerant streaming *data acquisition pipeline* in a distributed big data setup, mining, and searching patterns in it while data is still *in transit*. Data ingestion is the first layer in the proposed Multimodal Analytics Framework (MAF). As shown in

Figure 3.2 data collected from heterogeneous sources are processed before they are persisted into a big data cluster (Hadoop or Spark). Stream data ingestion is augmented with a multi-criteria decision-making framework discussed in Chapter 4. The data ingestion component in MAF is validated using a case study on a big data real-time clickstream data ingestion pipeline. The technical stack is developed on top of the standard big data tools such as Kafka, Flume, Spark and Cassandra.

Big data real-time data sources include both machine sourced data such as networks, IoT, servers, sensors, legacy systems, and user-generated content posted by the users in the online platforms. The thesis deals with user-generated content primarily from e-commerce systems as well as machine data like patient-generated health information (detailed in Chapter 5). This chapter describes the methods for ingesting data from user-generated content from web content. The methods can be well extended for other types of data sources.

Websites generate a large volume of real-time data through users' *clicks*. Web portals track the users' browsing patterns simultaneously in real-time and in batch mode. Mining *browsing motifs* to display *personalized recommendations*, and near-real-time tracking of *recently viewed products* greatly enhances overall user experience and helps generate revenue. Users' *click to view* items accumulate a large data volume compared to a tiny percentage of clicks converted to final checkouts. Near real-time processing of a large data pool for creating unique personalized, contextual experiences need quick analysis of the inflow of data before even being stored into a database of records. Coupled with zero tolerance for data loss, the challenge gets even more daunting. In a real-time setting, instead of waiting for data to be gathered in its totality, at a long periodic batch interval, the streaming analysis leads us to detect patterns and draw conclusions based on them as data start arriving. Big data streaming is the data processing paradigm designed with infinite datasets in mind. Introduced as a new category of open source project, a scalable stream processing paradigm, by Nathan Marz, creator for *Apache storm* [276] while developing an ingestion pipeline for Twitter. Apache Storm, Spark Streaming, Apache Flink are some of the popular real-time distributed processing frameworks allowing users in-flight processing on the inflow of data before data is even stored in the database [55, 119].

The data ingestion techniques presented in this chapter are validated using a case study analyzing and predicting user *clicks* in real-time through a machine learning approach on a huge volume of heterogeneous data pool that requires a new model for stream processing frameworks. In the storage layer, NoSQL datastores deal with trade-offs between Consistency, Availability, and Partition Tolerance (CAP) and also between throughput, latency, and correctness. This chapter also describes a near real-time data storage, and processing approaches to analyze streams of click data implemented on Apache Storm and Cassandra NoSQL datastore to bring insights into consumers' browsing motifs and build the recently viewed products list at near equal to the pace users continue to click on different links.

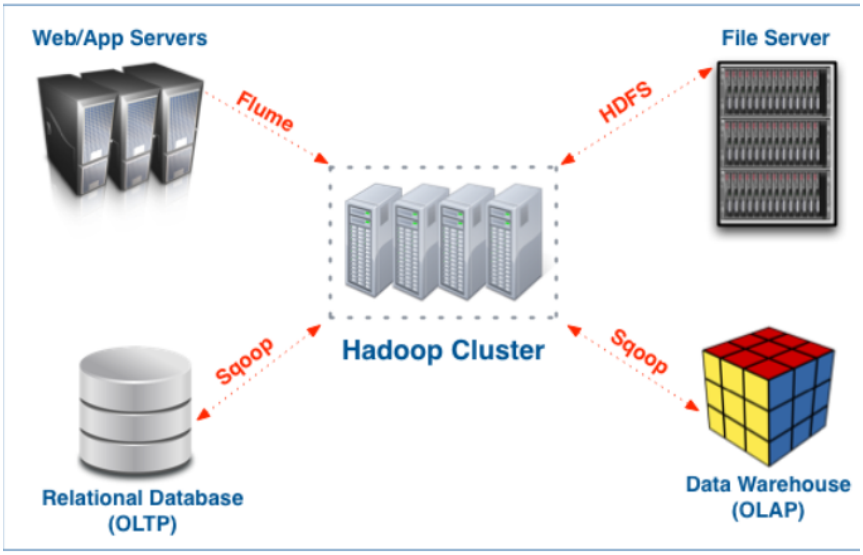


Figure 3.2: Data collected from heterogeneous sources before persisted into a Hadoop cluster through data ingestion tools like Flume or Sqoop. Haddop Distributed File System (HDFS) is the most common data storage option.

3.2 Organization of the chapter

This chapter begins with a review of several existing big data real-time distributed ingestion frameworks like Kafka, Flume, and REST Web Services in Sections 3.4 to 3.6. Lambda Architecture

which is used for data ingestion and processing, is described in Section 3.7. In subsequent sections, several data ingestion mechanisms are implemented through a case study of clickstream data capture and in-flight processing. Section 3.8 provides the usefulness of the proposed framework in the clickstream analysis. The sections 3.9 and 3.10 presents data retrieval techniques in real-time through users' clicks in a browser, mining data-in-transit before storing into a big data storage. Section 3.11 outlines the strategy to minimize database latency, trivial to achieving real-time turnaround time. A case study for real-time clickstream data analysis using Apache Storm is presented in Section 3.12. The application is hosted in the Microsoft Azure Cloud environment (Section 3.13).

3.3 Summary of Contributions

The primary contributions of the chapter are the new methods for stream data ingestion, in-flight processing and distributed storage, as summarized below:

3.3.1 A Novel Fault-tolerant Ingestion Component for Distributed Big Data Infrastructure

- This work presents a data ingestion component as part of the proposed MAF. Data ingestion takes place in the streaming layer through a host of big data tools and services such as Kafka, Flume, Spark, and Cassandra.
- A new method describes the processing of *late-arriving data* on the *event generation time* through a heuristic watermarking concept that adjusts the threshold time dynamically to wait for the late data.
- Introduces an event *chaining scheme* to aggregate click events from multiple simultaneous sources for a consolidated view.
- Presents a horizontally scalable big data framework for ingestion to processing, hosted on

the cloud environment.

3.3.2 Distributed Storage Layer

- The chapter introduces a stream data storage using Cassandra datastore to support a low-latency, highly available stream processing architecture.
- Proposed theorems describe trade-off between *low-latency* and *high-accuracy*.
- The chapter provides unique insight into optimizing *Cassandra* database on a multi data centre setup for near real-time responses.
- Load test results are provided using the *Datastax Enterprise* (DSE) distribution of Cassandra.
- Experimental results are presented for clickstream data flow from *Kafka* to *Cassandra*.

3.3.3 A Case Study Through Clickstream Data Analysis

1. The case study is based on the big data *Lambda Architecture (LA)* [103], which combines both batch and stream processing approaches. LA is used to analyze users' click data on e-commerce sites. In the real-time setting, the model builds the *recently viewed products* list, which a user previously viewed. At the batch setting, the model simultaneously analyzes *motifs of users* for computation of *personalized recommendations*.
2. This chapter introduces novel techniques in clickstream data analytics to unleash key customer journeys through pattern mining using the n-grams and Student T-Test, which distinguishes between regular patterns and special sequences [225, 297]. A model is proposed to predict users' transition from one state to another based on the higher-order Markov chains.
3. Clustering clickstream sequences help generate customer segments and build personalized recommendations. A Custom-built model consolidates both batch and real-time data pools. First, the initial cluster is formed through the batch setting. Thereafter, at a streaming setting,

a variation of the Expectation-Maximization algorithm with Gaussian Mixture Model maps each user click (stream event) to one of the clusters at every streaming window interval. A dimension reduction technique reduces training time significantly.

4. A model for the Apache Storm topology is presented serving near real-time responses.

3.4 Big Data Infrastructure for Capturing Clickstream Data

This work identifies two distinct approaches for capturing real-time clickstream data.

1. *Approach 1:* The straightforward approach is, collecting data such as web browser log files containing user's browsing data and ingest them into storage such as HDFS.

    - *Disadvantage:* No real-time processing. The method is inherently batch-oriented. As the Hadoop/Spark batch processing would require data accessed from HDFS/NoSQL for further processing, adding a further delay to the responses. As the raw clickstream events are unstructured, direct data insertion require additional parsing, cleansing, and re-inserting into a datastore.

2. *Approach 2:* This method does not rely on server-side logs processed in batches. Rather, an event is generated on the client-side browser and transferred to a separate back-end service to handle the event processing and logging while data are still in transit, without the need for storing.

    - *Advantage:* The method allows real-time processing and enables custom events created from JavaScript processed in-flight before ingesting into HDFS. The technique enables processing responses in real-time or near real-time. This work adopts *approach 2* for the data ingestion frameworks.

The following section describes stream events that are ingested through distributed big data ingestion frameworks such as Kafka and Flume in accordance with *approach 2*. Data propagates

through Kafka or Flume, which are distributed message queues, and subsequently processed by Storm or Spark before being stored into a Cassandra datastore.

3.5 Data Ingestion Using Kafka

Ingesting large volumes of high-velocity data requires fast, fault-tolerant, distributed pipelines. Apache Kafka, a massively distributed client-server-oriented publisher-subscriber messaging system, replaced many traditional message queue systems like Rabbit MQ, IBM MQ because of its higher throughput, reliability, and replication capability [116]. In the current framework, Kafka is the central hub for real-time processing in integration with Spark streaming APIs.

The following big data tools are used along with Kafka for data ingestion:

- **Apache Spark:** Spark subscribes from Apache Kafka stream as a data source and performs real-time processing prior to saving incoming stream into a datastore.
- **Apache Cassandra:** Processed data is persisted into Cassandra, a columnar NoSQL database, for further batch processing.

The end-to-end flow is depicted in Figure 3.21.

A data ingestion simulation generates a high rate of click data and subsequently writes into a Kafka topic. Spark is a consumer to the Kafka stream producers that analyzes the stream and persists the results into Cassandra. The data flow pipeline is described in Algorithm 1:

3.5.1 Experimental Setup with Kafka

This section discussed the data flow pipeline using Kafka, followed by processing by Spark and storing it into Cassandra. The setup is hosted in Amazon Cloud Services (AWS). The cluster uses Amazon AWS medium level server configuration, as shown in Table 3.1. First, a Kafka messaging queue is built capable of handling streaming text (clickstream) using a string serializer/deserializer. Spark is a consumer to Kafka queue and processes input stream by counting the number of events

Algorithm 1 Live Data Ingestion and processing using Kafka and Spark

Input: Stream data, Kafka broker address, Cassandra server address
Output: Processed data saved into Cassandra datastore
Initialize: topic, broker, Cassandra Consistency level, Cassandra server
Create a Kafka stream object with the broker configuration
for every batch window interval **do**
  Subscribe Kafka from Spark Stream
  if (data requires cleansing) **then**
    Filter and transform data
  end if
  Spark consumes Kafka queue, processes and commits to Cassandra
end for
return Total message count in each window

at each passing window interval of *one* minute. Once the streaming job is started, Spark continually processes and ingests data into the Cassandra database every *one* minute at a pre-created keyspace and column family. See Algorithm 1.

Table 3.1: AWS Instance Types

Instance Type	vCPUs	Memory	Instance Storage	EBS Optimized Bandwidth
m1.large	2	7.5 GB	500 GB	Moderate

3.5.2 Observations

Spark streaming processes the upstream data from the Kafka queue that is connected to the data source. In the processing layer, Spark *parses* each record from the real-time incoming stream and counts the number of events on each window interval. Experimental results are provided for stream processing throughput, scheduling delay and processing latency as shown in Figures 3.3, 3.5, 3.4 and 3.6. The write throughput for Kafka, Spark, and Cassandra are measured by scaling up the cluster size horizontally with an added number of nodes in the cluster as shown in Figure 3.7. Cassandra consistency level is kept to *one*, which is the lowest level of consistency setting to

achieve maximum write performance possible. The test was run for 11 hours 54 minutes (plotted along the x-axis), and the right-hand side figures show the data distribution (histogram).

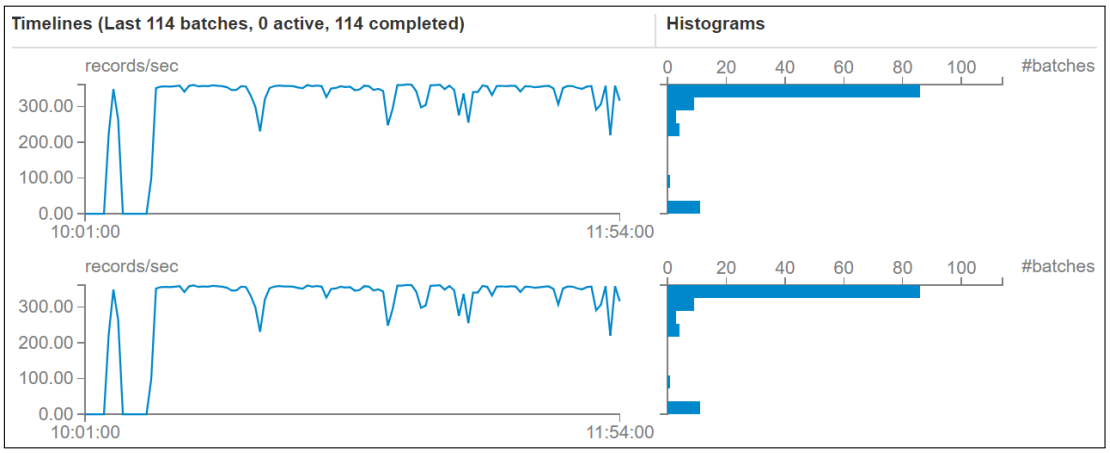


Figure 3.3: Input Rate for the Kafka to Spark ingestion is 306.25 records/sec (avg). The input Rate is the speed at which data is ingested from upstream data sources.

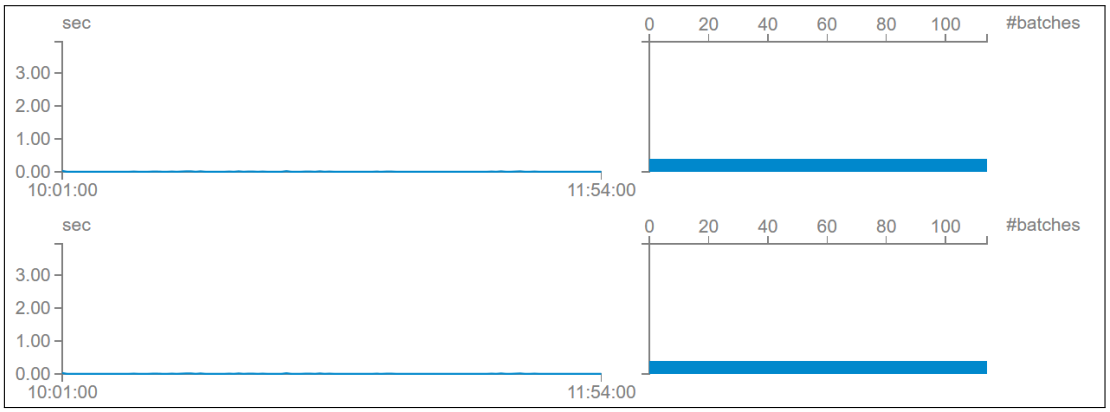


Figure 3.4: Scheduling delay for the Kafka (ingestion) to Spark (processing) is 6 ms (avg). Scheduling delay is the time spent from when streaming jobs in mini-batches was submitted to the time when the first streaming job was actually started (out of possibly many streaming jobs waiting in a queue).

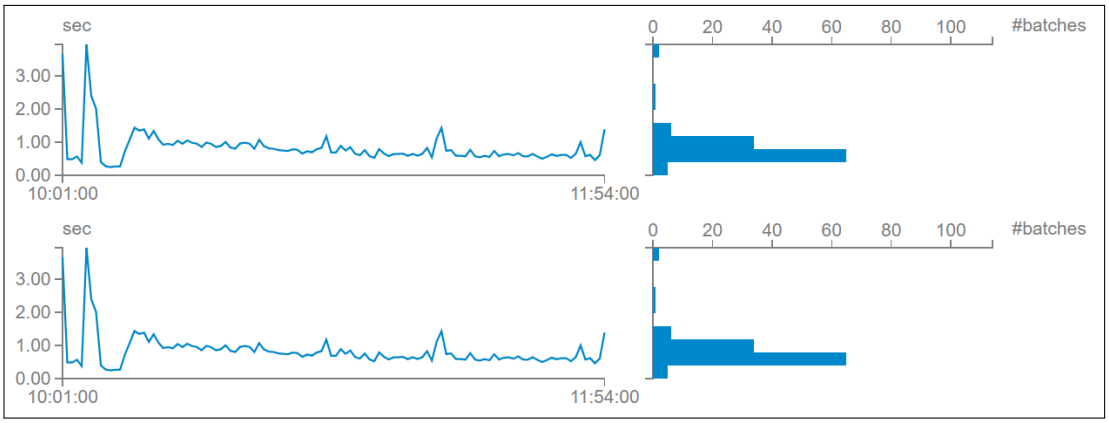


Figure 3.5: Processing time for the Kafka ingestion to Spark processing is 833 ms (avg). Processing time is the duration spent to complete all the streaming jobs of a window.

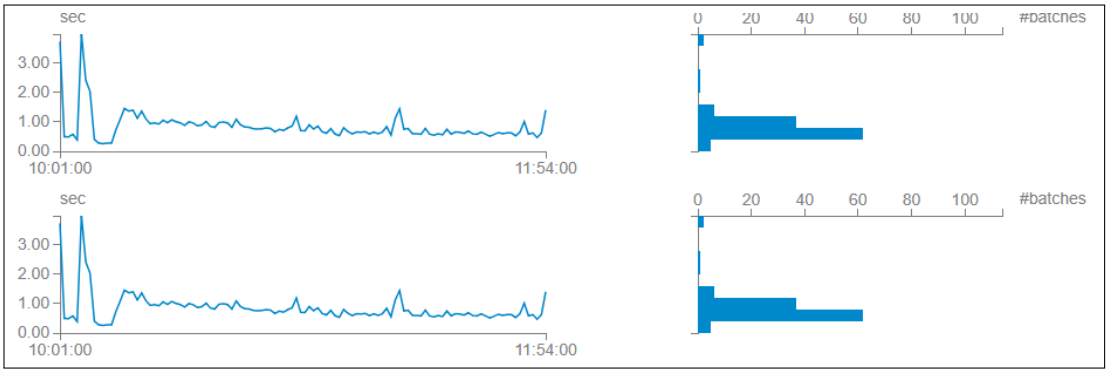


Figure 3.6: Total delay for the Kafka (ingestion) to Spark (processing) is 839 ms (Avg). Total delay is the time spent from submitting to complete all jobs of an streaming window.

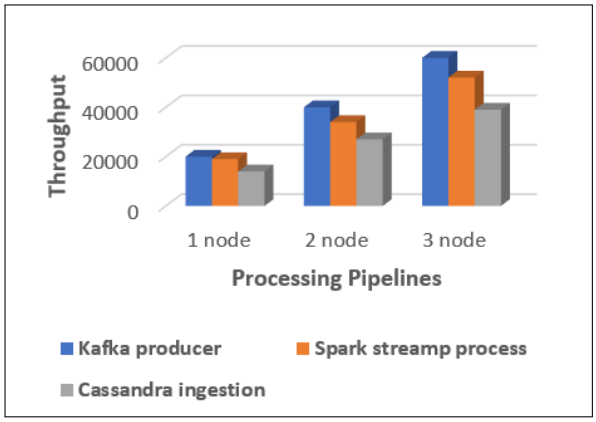


Figure 3.7: Write throughput vs increasing cluster size when data moving from Kafka to Cassandra. Lag in the throughput down the processing pipeline from Kafka to Spark and finally to Cassandra is due to network latency and processing delays in the intermediate steps.

3.5.3 Watermarks: Processing Late Arriving Events

In the event of time windowing, the data ingestion framework takes a finite set of data according to the time they occurred. Most of the older versions of stream processing systems do not consider the event generation time. If the events arrive late, the application simply considers the processing time. Accepting *processing time* as the *event generation time* can lead to major errors in the analytical insights as time-series events needs to be processed based on generation time. To consider the processing time for the late-arriving events, the processing engine should maintain a state and allow late data to update the window state until a threshold time. The threshold is defined by watermarks to manage how long to wait for the late events.

In real-world distributed data source scenarios, it is computationally infeasible to provide an estimate of data arriving delays. Data transportation over the network or processing bottleneck at multiple stages takes an unpredictable time. To address the latency issue, several approaches have been considered. An adaptive SLA-based data flow mechanism for stream processing engines was explored [99]. In the current framework, a heuristic *Watermark* is proposed that adjusts the threshold time to wait for the late data. Observe from Figure 3.7, when the data stream moves from Kafka to Spark, there is a significant drop in the number of events processed between the

transition from Kafka to Spark. Unprocessed data remains buffered in Kafka until the value for *time-to-live* or *data size* reaches the maximum threshold. Since the number of unprocessed data remains unpredictable at all times, a heuristic value is assigned to *watermark* on the following criteria:

- By tracking the minimum event time of unprocessed data waiting in the queue.
- Monitoring the growth rate of unprocessed data in the queue.

Below is a code snippet (in Spark and Scala) defining the *Watermark* of the query on the value of the column *timestamp*, and x is the *heuristic threshold* value:

```
clickstream.withWatermark(timestamp, x)
```

3.6 Data Ingestion through Flume

Moving data on a real-time basis over the internet can cause the network to be busy at all times. If a millisecond range response is not required, then batch processing can be opted using tools such as Apache Flume. Also, moving a small amount of real-time data can be particularly difficult for HDFS storage since Hadoop is designed for large files. Therefore, with data sources with a large number of small files, Flume agents can collate multiple files and flush data in a batch mode as a single large file.

Consider scenarios where click-data is simultaneously generated from multiple sources. A common example is logs collected from hundreds of web servers sent to multiple collector agents writing to the HDFS cluster. Each collector agent is autonomous and specialized in a particular domain of operation. Apache Flume *agent-chaining* is used to combine data from multiple sources for consolidations. The chaining of agents enables aggregation or propagation of event data in a modular way. As shown in Figure 3.8, Flume *agent-chaining* contains three components: source, sink, and channel.

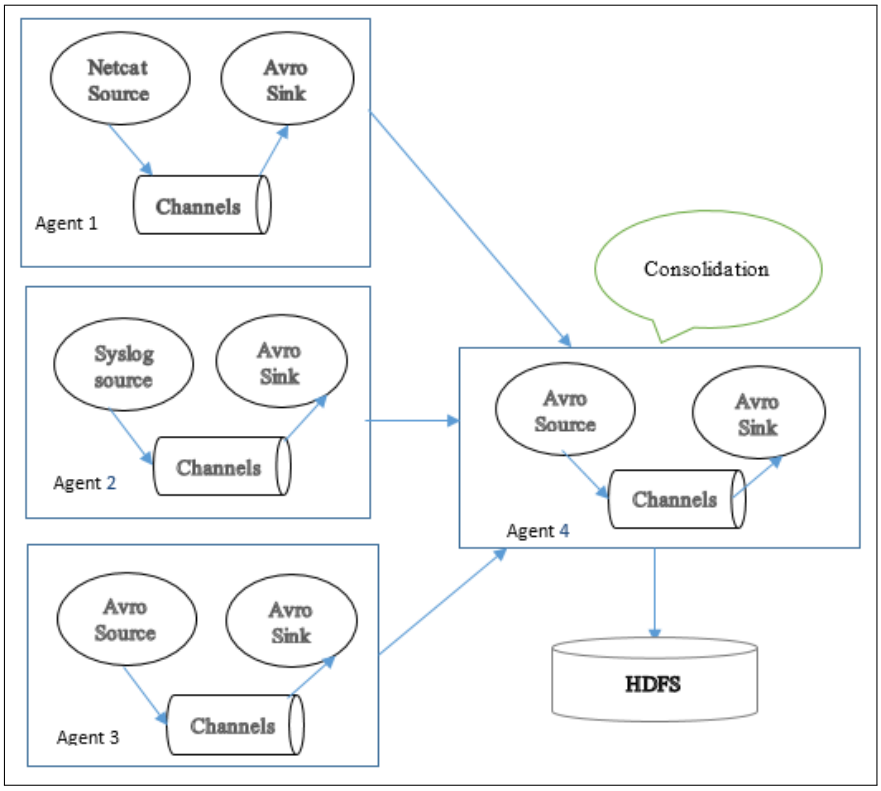


Figure 3.8: Flume chaining of events. Event chaining aggregates click events from multiple simultaneous sources resulting a consolidated view.

Source: A source is the component of an agent receiving information from the data generators and transfers it to one or more channels in the form of Flume events. Apache Flume supports several types of sources, and each source receives events from a specified data generator. Avro, Thrift, Netcat, etc., are some of the standard sources for Flume.

Channel: A channel is a transient store that receives the events from the source and buffers them till they are consumed by *sinks*. The channel is a bridge between the *sources* and the *sinks*. The channels are fully transactional, and they can work with any number of sources and sinks. Examples of channels are Java Database Connectivity (JDBC) channel, file system channel, memory channel, etc.

Sink: A *sink* stores the data into centralized stores like HBase and HDFS. The *sink* consumes the data (events) from the channels and delivers it to the destination. HDFS or any other datastore can be a Flume sink. A Flume sink can forward the stream to another Flume source of the next Flume agent (next hop) in the flow. An example is described below.

3.6.1 Experimental Setup with Flume

An experiment is carried out to benchmark the real-time data ingestion throughput, i.e. the amount of data ingested per second, using Flume. Consider a setting where Flume consumes incoming stream in a directory as new files are added to the directory. The data source is created as a *spool directory* that lets data generators such as a *server* or a *sensor* ingest small files as soon they are created in real-time. Flume source continuously tracks the directory and fetches all new files in the spool directory after a configurable data batch size is reached. Instead of transferring each small file as they appear, Flume combines all the incoming files into one large size file and moves to the destination, i.e. a *Flume sink* [224].

The system configuration for each node in the Amazon Cloud Platform is described in Table 3.1. Single node Flume installation was used with *two* nodes Hadoop and Hive installation. Flume used *five* sources with event size of 400 kb each.

3.6.2 Observations

Since Flume does not support distributed architecture (client-server or peer to peer), a single Flume agent cannot be distributed across multiple nodes. However, multiple agents can be created to load-distribute the overall input stream data. Figure 3.9 shows the experimental results of a single agent flume client. Write throughput largely depends on a sink type (HDFS, Hive etc.) and is proportional to the level of parallelism (number of sources and sinks).

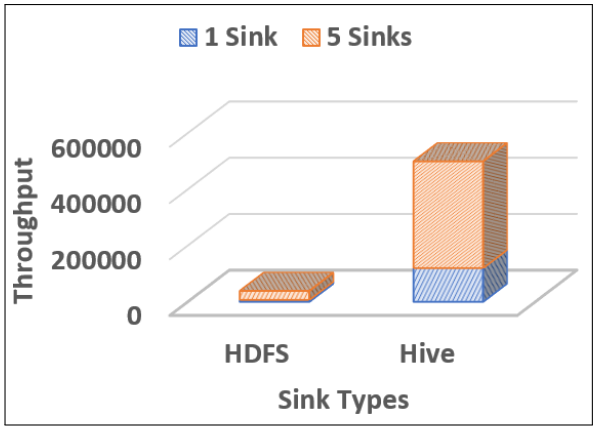


Figure 3.9: Flume throughput on different sinks. Increasing the number of sinks increases the parallelism level and throughput.

3.6.3 Ingesting data through RESTful Web Services

Web Service provides a great way of interoperability between software applications running on heterogeneous platforms and frameworks. In terms of big data information retrieval, the UI layer generates a clickstream on each user click, and the datastore persisting user clicks are hosted on a different cluster for efficient load distribution. UI and datastore systems are more often installed on different operating systems, and web services can effectively bind them together. RESTful web services can provide such interoperability for ingesting data from clients' browsers to a centralized database [233].

A Kafka producer or Flume source that binds to a data generator are configured to continuously

listen to the source system, and port number RESTful Web Services are writing to. Restful Web Services send and receive messages using JSON or XML. This makes the big data development more *decoupled* as the data ingestion layer can be developed and tested individually. *Advanced REST Client*, is a Chrome browser extension that simulates the clickstream events generation through JSON formatted data. A *POST call* screenshot is shown in the Figure 3.10.

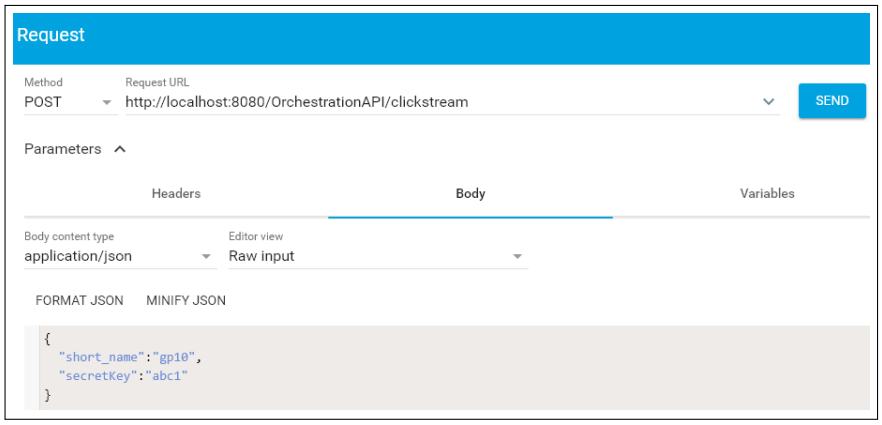


Figure 3.10: RESTful Web Service requests through Advanced REST Client interface. Browser extension client utilities help simulating click stream over HTTP requests.

3.7 Lambda Architecture for Concurrent and Mix Processing

The Lambda Architecture [104] is a concurrent mix-processing paradigm, which combines high throughput Hadoop batch setup with low-latency real-time frameworks over a large distributed environment. Lambda Architecture provides a *simultaneous* and *mix-processing* environment where ingested data is dispatched to both the *batch* and the stream layer. The *stream* layer serves only low-latency interactive queries, and the batch layer serving the high-latency jobs with higher accuracy. Processing results from both the layers are merged for the type of queries that require comprehensive historical data with the most recent updates. See Figure 3.11.

A case study presented later in this chapter using Lambda Architecture to build recently viewed product lists at a near equal pace as users continue to click on website links. Apache Storm stream

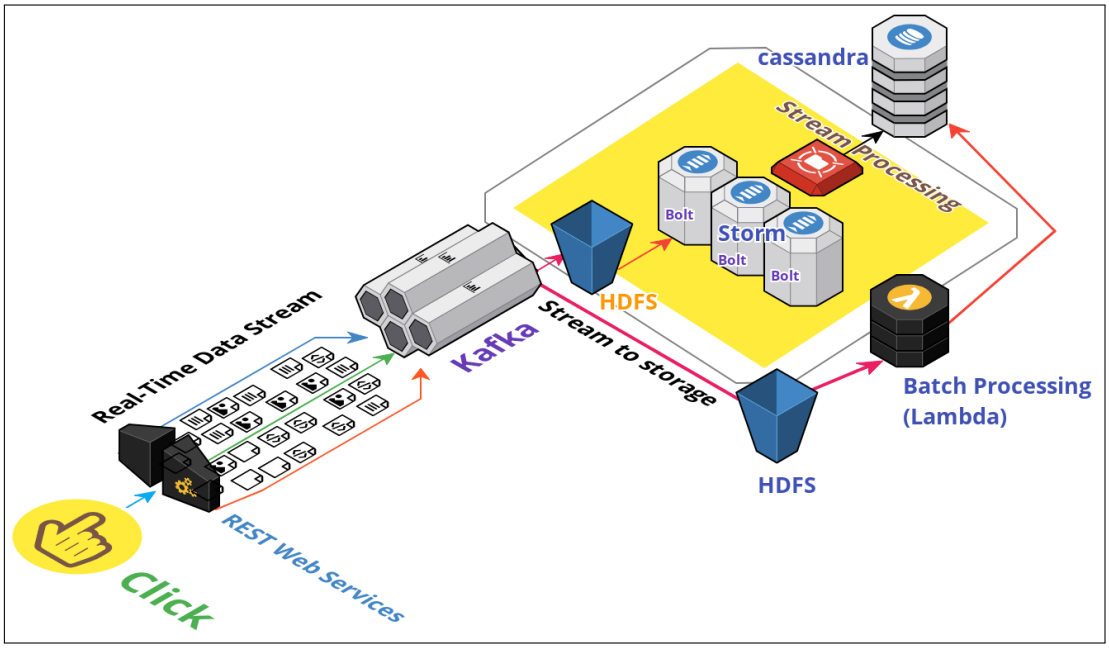


Figure 3.11: The architecture is showing the end-to-end flow of real-time data processing in combination with the batch. Lambda Architecture blends a high throughput Hadoop batch framework with low latency real-time frameworks over a large distributed setup. Note, while data is processed in real-time as Storm Spouts and Bolts, concurrent processing with the batch setup is carried out using the historical dataset out of HDFS. Cassandra combines the views from the stream and batch.

processing pipeline cleanses and transforms data to store into Cassandra DB. The recently viewed list is built by a low-latency Cassandra query as users continue to click on different items. Java Spring REST APIs return JSON responses to the user interface to display the *recently viewed* item list. At the same time, a simultaneous data pipeline stores the data into HDFS, and a periodic batch job, using Apache Hive, performs data mining to extract user *motifs* through *n-grams* and build a recommendation service by *Collaborative Filtering* [187]. Section 3.10 elaborates on the case study further.

3.8 Clickstream Analysis Application Scenarios

In this section, the data ingestion method is illustrated through a case study of clickstream data capture and in-flight processing. Starting with the *usefulness* of the clickstream analysis, the following sections present real-time data retrieval from the client browser on each user's clicks, mining data *in-transit* before storing into a big data storage.

Why analyze the clickstream data:

1. **User experience optimization:** Clickstream helps understand the user's mindset and improve sales through better engagement. E-commerce sites collect data tracing users trails to analyze the pages they are visiting most and in which order. Web traffic analysis reveals the path users follow while browsing. E-commerce marketers can improve on the overall users' experience by getting insights about the responsiveness of the website, how frequently users click the *back button* or the *stop button*, and the amount of data transmitted before users move on to the next page. Marketers can quantify the effectiveness of the sales volume. The analysis results show items added to the cart and removed and items purchased together. Powered by clickstream data analysis insights and market trends, marketers improve the click path by optimizing the site design and effectively reducing *bounce ratio* (items added to cart but not checked out) and improve *view* to *purchase* conversions.

2. **Building recently viewed products list:** The recently viewed products list enhances the

overall shopping experience by reminding users of products they have viewed before. Processing data near-real-time is the main technical challenge on a large volume of the data pool.

3. **Market driven analysis:** Basket analysis produces insights into the aggregate behaviour of customer buying patterns. Similar to car drivers who can use different routes to reach the same destination, analysis of a user basket uncovers the common interest across consumers, the common path users take to conclude a purchase. This leads the way to find out the most effective path a buyer takes to search for and buy a product.

4. **Analyzing the next best product:** Next Best Product Analysis (NBP) is essential for marketers to predict the next purchase by a customer. NBP analysis discovers customer buying patterns to list the items consumers tend to buy together. For instance, a customer who buys nuts *also-bought* bolts together. Once the *also-viewed* and *also purchased* behaviour is learned for a collective pool of historical users, a new customer can be offered a product together with another product he already bought. This helps to generate major revenue for the site since the analysis shows that consumers tend to buy more often from suggestions and recommendations than by their own search.

5. **Allocation of website resources:** Clickstream analysis uncovers the key browsing patterns which are used to distribute the resources (hardware and development time) to focus areas.

6. **Segmenting consumers at a micro-level:** High impact personalized recommendation is achieved through micro-level customer segmentation using clickstream data clustering. Customers are grouped based on buying pattern and average cart value, which helps to provide a targeted recommendation that users are most likely to buy.

3.9 Retrieving Ingested clickstream Data

Clickstream is recordings of users' click navigation trail while surfing a website [94]. The user action is captured in the client-side browser. Therefore, clickstream data is the URL generated when each user clicks on items on the site. For instance, in an online shopping portal, clickstream data may look like this:
http://xxx.com:8080/electronics/products?userID=id3&product Name=iPhone10&price=500 &location=london. Each context parameter like *product name*, *price* and *location* is appended to each clickstream events. Each time users open a new session, a new user context is captured. The context is a uniquely derived entity built from the session object created at the application server and JavaScript layer. A context ID is appended to each user's click data. The context ID is used for identifying each user uniquely and tracking their previous browsing patterns based on the system IP users browsed from. Thus, the website need not rely on users to explicitly log in to track their identity.

3.9.1 Identifying Unique Users through Context ID

E-commerce users can see their recently viewed products list even when they are not logged in. So, tracking each user's past interactions for a non-logged-in user requires identifying a unique user session from the application server's session ID. Every user of a site is linked with an application-server-generated *javax.servlet.http.HttpSession* object, which is persisted and fetched later for session information about that user. Consider that clicks from each unique IP belong to a unique user for a unique session. A *recently viewed items* list is built for a user based on a concept that essentially combines all the user's past sessions into one single list. This approach has the limitations of not being able to capture events correctly when multiple users are accessing the same IP or the same user browsing from different IPs. The limitation is avoidable, considering only logged-in users or using the session id. The final order of the recently viewed products list, sequence, and pattern extracted from the context ID is the key determining factor for analyzing *motifs of users* and computing *personalized recommendations*.

Before developing the method for analyzing clickstream sequences and understanding users' motifs, a few concepts and measuring techniques for quantifying user behavioural patterns and motifs are defined first. The topographic constructs of clickstream sequences are described, and an estimate of users' interests like the support of a sequence, common patterns, and motifs are formally defined.

Definition 3.9.1. ***Set of Unique Sequences.*** *A set of ordered and unique sequences S_i represents user activities under a single browsing session persisted in a database of tables.*

Definition 3.9.2. ***Subsequence.*** *A: $a_1, a_2, ..., a_n$ is a subsequence of B: $b_1, b_2, ..., b_n$ if and only if there exists $k_1, k_2, ..., k_m$ such that $1 \leq k_1 < k_2 ... < k_m \leq n$ and $a_1 \subseteq b_{k1}, a_2 \subseteq b_{k2}, ..., a_m \subseteq b_{km}$*

Definition 3.9.3. ***Support of a Sequence.*** *Support of a sequence S_i is the count of sequences in database D which has S_i as a subsequence. If the count of matches (support) for S_i in the database is more than a defined threshold, then S_i is called a frequent sequence [194].*

Definition 3.9.4. ***Set of Unique Events:*** *A set of unique ordered click events E_i combines to form a sequence S_i, such that $\sum_{i=1}^{n} E_i = S_i$*

Definition 3.9.5. ***Set of Common Patterns:*** *A set of common patterns P_i is identified as set of multiple sequences or subset of a sequence. Hence, a pattern P_i is defined as $P_i \subseteq \sum_{i=1}^{n} S_i$ or $S_i \subseteq \sum_{i=1}^{n} P_i$*

Definition 3.9.6. ***Finding the Motifs:*** *A user's motif (M_k) to use the site is a combination of patterns represented as n-grams with n representing the count of patterns such that $\sum_{i=1}^{n} P_i = M_k$.*

The following section explains the steps for understanding user motifs through pattern mining. Hadoop ecosystem tools such as Hive has good support for native n-grams APIs, which are used to highlight commonly occurring sequences that appear more than a defined threshold as defined in Definition 3.9.3 and 3.9.5. Understanding user behaviour through n-grams is a high latency task handled through periodic batch jobs.

3.9.2 Browsing Hierarchy

Users' browsing patterns are grouped into categories based on the browsing path. The parent category is divided into several subcategories. For example, the category *electronics* is sub-categorized as *television* and further sub-categorized as *3D television*. A custom JavaScript is added to the *header* of each UI JSP page, captures the users' navigation patterns in terms of each click. The navigation trail creates the browsing path for each user for each session. For instance, users first open the home page *http://a* then navigate to the subcategory *b* at the URL *http://a/b* then come back to the home page again. The clickstream pattern for the user U is: a, b, a. Each time the user navigates to a new category and subcategory to the leaf of the tree, a new click event is captured. If a user closes the session by closing the browser and opens the home page again, the new session sequences are not included in the old pattern. Items browsed under the same browsing session are the key to understanding the user motifs and building the recommendations.

Total Time Spent Per Subcategory

Along with browsing frequencies, total time spent per user per subcategory on each session reveals user browsing patterns. Time spent for each item under a subcategory are added. A parent category (like *electronics*) can be too generic to be meaningful for user segmentation. Therefore, subcategory (like *television*) is used for computing total time spent. Let d_i^j be the time spent on a subcategory c_j by a user u_i then $d_i^j = \sum_{i,j=1}^{n} duration(u_i, c_j)$

Table 3.2: Clickstream Sequence Table

Session 1 User A	**Session 2 User A**	**Session 1 User B**	**Session 1 User C**
http://a	http://a	http://e	http://a
http://a/b	http://a/c	http://e/f	http://a/b
http://a	http://a/c/d		

The sequence of events captured by the system is as follows:

Table 3.3: Item Sequence Table 1

User ID	Sequence ID	Event Time Stamp	Leaf Element of the Path
U_1	1	10	a
U_1	1	20	b
U_1	2	30	b
U_1	2	40	c
U_1	2	50	d
U_2	3	20	e
U_2	3	40	f
U_3	1	10	a
U_3	1	20	b

Table 3.4: Item Sequence Table 2

User	Session ID	Elements	Size	Root Category
U_1	1	[a,b]	2	a
U_1	2	[a,c,d]	3	a
U_2	3	[e,f]	2	e
U_3	4	[a,b]	2	a

Per user per element frequency is computed using Equation 3.1

$$freq_u^i = count(user_u, item_i) \tag{3.1}$$

Frequency of co-occured items i and j is given by

$$freq^{ij} = count(user_u, item_i, item_j) \tag{3.2}$$

The relative frequency for item i and user u =

$$\frac{count\ of\ visits\ to\ item\ i\ by\ user\ u}{Total\ count\ of\ visits\ by\ user\ u}$$

$$RelativeFreq_u^i = \frac{count(user_u^i)}{count(user_u)} \tag{3.3}$$

A co-occurrence matrix M_{ij} is built between two items (i and j) across all of the user base. The matrix is the aggregated count of items *co-viewed* under the same browsing session.

Table 3.5: Co-occurrence Matrix

Item Viewed	Item Also Viewed	Count
a	b	2
b	c	1
c	d	1
e	f	1

$$M_{ij} = count(item_i, item_j) \tag{3.4}$$

3.10 Understanding Users' Motif Through Pattern Mining

The sequence of events that co-occurred more than average reveals an important user trait. For understanding browsing pattern, *n-grams* are computed with n ranges between 2 to 7. *n-gram* cap-

tures the frequency of a contiguous sequence of *n* elements from a corpus of text. The sequences are sorted based on the pattern length and occurrence frequency to reveal the most popular webpages and the sequences a user is reaching the final webpages. Apache Hive APIs [4] can be used to compute n-grams and find the most commonly occurring sequences and their counts.

n-grams measure the popularity of a webpage sequence but do not determine if the *mean* of two sets of data are significantly different from one another. Consider a table of data with two columns as browsing sequence (category and subcategory) and an average price of a subcategory; divide the table into two, one with a price of more than $25 and the other with a price less than or equal to $25. To estimate the difference in the data distribution of the two price groups *Independent two-sample Student's t-test* is used as a *Collocation* method [163]. Experiments use the production dataset obtained from data.world [5] [6] consists of data from Amazon.com fashion products and Indian e-commerce major Flipkart's product details. Table 3.6 shows the results for the t-test with *Amazon* and the *Flipcart dataset*. *Flipcart dataset* contains raw clickstream events in the *product_url* column along with price and other information. Amazon dataset captures the clickstream for products *also bought* along with the name of the product *bought*. *Amazon dataset* is compared for two distributions separated based on the price (price more or less than $25). *Flipcart dataset* is split into two and compared for browsing patterns in the odd and the even months

Table 3.6: Student's T-Test

Returned Parameters	**Amazon Dataset**	**Flipkart Dataset**
h-value	1	0
p-value	$3.751e^{-07}$	0.1098

The return values [h,p] signify if two distributions are substantially dissimilar from each other at a 1% significance level. With $h = 1$ in the *Amazon* dataset, the browsing pattern for two distributions of different price ranges but with similar click sequences are significantly different. On the other hand, with the *Flipkart* dataset, considering the click sequences, clickstream events for odd and even months are not much different.

3.10.1 Clustering Clickstream Sequences

Users take different paths while browsing the site. Permutations and combinations of click paths make clickstream analysis and prediction data intensive. Grouping the clickstream data helps create customer segments with users of similar interests. Clustering can be used for future click predictions for similar users.

Clickstream is clustered based upon user browsing sequences under a session, browsing frequencies on the same path and duration spent on each subcategory. Consider two clustering approaches for customer segmentation and prediction: a hybrid model of K-Means clustering and the Expectation-Maximization algorithm with Gaussian Mixture Model. Initial clusters are created by Apache Spark APIs K-means clustering under the batch settings for the historical dataset as a one-time job. Thereafter, at each streaming window interval, Apache Spark Streaming K-means APIs map each user click event to one of the clusters and updates the cluster incrementally at a near-real-time setting [189] [191] [7]. However, K-means *hard assigns* an event to a specific cluster, limiting the probability of customers belonging to multiple user segments with varying degrees of possibility. Therefore, we can take recourse to a variation of the Expectation-Maximization (EM) algorithm with the Gaussian Mixture Model (GMM) at the streaming setting. EM *soft allocates* events to distributions with a probability of any event belonging to a distribution mean, resulting in users falling into multiple customer segments with varied degrees of association. The predicted cluster is used to *suggesting* the user a set of items under the same browsing session. For instance, if the user views items *a, b, c* under a browsing session, then based on the sequences on the existing clusters that are similar to the user's browsing pattern, items *d, e, f* can be suggested to a user. The challenge is, since users expect to see the suggested items under the same browsing session, as they continue to click different items, the computation needs to return near-real-time results. As the browsing data volume is large on even a relatively moderate user base, a real-time big data ingestion and computation framework are essential.

3.10.2 Expectation Maximization (EM) Algorithm with Gaussian Mixture Model

Let us assume c^t_{ut} is the interaction between the user u and item i at time t. Then c^t_{ui} is given by:

$$c^t_{ui} = \omega p^t_{ui} + p^t_{ui} \tag{3.5}$$

where p^t_{ui} is the feature vector for the pair of user u and item i in a real-time setting. Vector p^t_{ui} is learnt through the historical batch dataset to provide an offset (or starting point) for the streaming process. p^t_{ui} is generally high dimensional and typically sparse. Standard learning methods [113] [193] are used to overcome the sparsity problem. Let us assume p^t_{ui} belongs to a $m \times n$ dimensional projection matrix $M_{m\times n}$. M is high dimensional and largely remains unchanged with very few cells requiring update at real-time. That is, $p^t_{ui} = M\delta^t_{ui}$. Only δ^t_{ui} is learnt at real-time which is not high-dimensional and the online model converges much faster. ω is the data sample row vector representing the weight of each of the events summing to *one*. The feature vector p^t_{ui} between user u and item i is a data frame with *three* variables as browsing sequence, frequency and duration in each sequence per customer. See an example in Table 3.7.

Table 3.7: User Browsing pattern

Weight	Customer ID	Browsing Element	Browsing Frequency	Browsing Duration
0.1	C1	A	10	20
0.6	C1	A-B	8	90
0.3	C1	A-B-C	400	120
0.1	C2	A	12	20
0.6	C2	A-B	15	20

The proportion of weight is pre-estimated and defined in the weight vector ω. In the Table 3.7, A is the parent category (example *electronics*) and given less weight than the more specific sub-category (example *television*) which reveals more about customer browsing intent. C and D are the leaf nodes representing a specific item with brand and model number (example: Samsung *television* model number UE50RU7100). Since Leaf nodes are too specific, they are given less

weight to avoid model overfitting.

The E-step

The expectation step aims to compute the *responsibility* for each data point for each cluster distribution. Let us create *50* customer segmentations within *one million* records in a $1million \times 50$ matrix. Expectation-step finds each data point's level of association to each distribution is given by Equation 3.6:

$$a_{ic} = \frac{\omega_c P(x_i|\mu_c, \sigma_c)}{\sum_{j\in(0,n)} \omega_n P(x_i|\mu_n, \sigma_n)} \tag{3.6}$$

where a_{ic} is association of the event x_i in terms of Gaussian distribution c. Total number of possible Gaussian distributions is n. The numerator represents the probability of event x_i for Gaussian distribution c multiplied by the weighting factor for each distribution. The denominator represents the sum of probabilities for n Gaussians for the event x_i.

The M-Step

With the degree of association value for each event for each Gaussian, we can improve the mean and standard deviation using the association values. The mean is calculated as

$$\mu_c = \frac{\sum_{i=1}^{n} a_{ic} x_i}{\sum_{i=1}^{n} a_{ic}} \tag{3.7}$$

Standard deviation is given by

$$\sigma_c = \frac{\sum_{i=1}^{n} a_{ic}(x_i - \mu_c)^2}{\sum_{i=1}^{n} a_{ic}} \tag{3.8}$$

Weighting factor ω_c is given by

$$\omega_c = \frac{\sum_{i=1}^{n} a_{ic}}{p} \tag{3.9}$$

where p is the total number of data points. That is, the sum of level of association values for each

of the Gaussian to an event divided by number of points.

3.10.3 Predicting User Click

A common case study in clickstream data analysis is predicting a user's next click sequences or the last click event. The prediction could help to suggest the users' list of items as recommendations. We can use Markov chains to predict users' transition from one state to another. Markov chains were first introduced by the 19^{th} century Russian mathematician Andry Markov [87] [181]. A Markov chain defines a stochastic process that determines the probability of the next events based upon the previous transitions and states achieved. In a Markov chain with k order, the future state relies on the previous k states. Formally defined as:

$$\begin{aligned} &\mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, ..., X_1 = x_1) \\ &= \mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, ..., X_{n-k} = x_{n-k}) \\ &\qquad (n > k) \end{aligned} \tag{3.10}$$

Fitting into the Markov model, the probability distribution $X(n)$ of the n^{th} click is given by Raftery [200].

$$P^{(n)} = \sum_{i=1}^{k} \lambda_i M_i P^{(n-i)} \tag{3.11}$$

M_i is a $m \times n$ transition probability matrix and λ_i is the weight of each lag (time difference of two parts of a same stochastic process) in i, such that,

$$\sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0 \ \ \forall i \tag{3.12}$$

If the final *absorbing* state [220] (checkout or add to cart) is known, then we can use the information about the final state F to predict more precisely the next click sequence as given by Equation 3.13.

$$P^{(n)} = F \sum_{i=1}^{k} \lambda_i M_i P^{(n-i)} \tag{3.13}$$

based on that, we can now set out the steps to predict clicks based on the Markov Model:

Creating a transition matrix

We compute a bigram transition matrix as the transition frequency between *two* websites links (items) as shown in Table 3.8:

Table 3.8: Bigram Matrix

	item 1	**item 2**	**item 3**	**item 4**
item 1	30	40	20	10
item 2	40	30	10	20
item 3	30	20	30	20
item 3	30	20	30	20
item 4	50	10	30	10

To compute the transition probability, bigram values are normalized by the sum of each row as shown in Table 3.9.

Table 3.9: Transition Probability Matrix

	item 1	**item 2**	**item 3**	**item 4**
item 1	0.3	0.4	0.2	0.1
item 2	0.4	0.3	0.1	0.2
item 3	0.3	0.2	0.3	0.2
item 4	0.5	0.1	0.3	0.1

Compute the Markov transition diagram

Table 3.9 graphically represented as the transition diagram (Figure 3.12) with edges representing click probabilities from one item to another. The following properties are analyzed from the transition diagram:

1. **Periodicity**: If there exists a cycle.

2. **Irreducibility**: If the diagram is a complete graph (each vertex is connected to another by

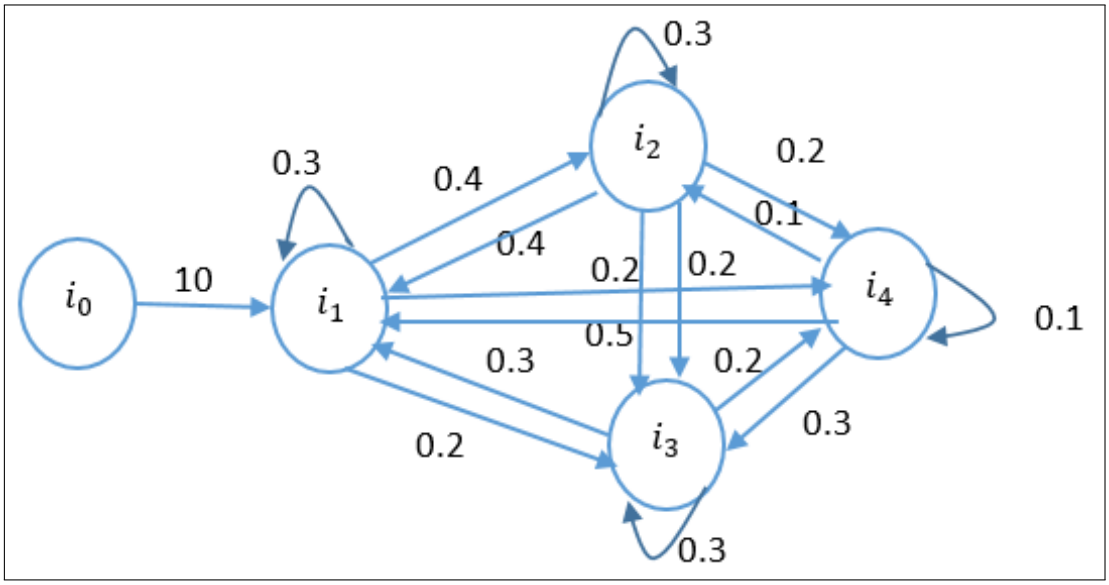


Figure 3.12: Transition diagram.

an edge).

3. **Invariant Distribution**: If there exists a unique invariant distribution (given the Convergence Theorem [176]).

   Considering Figure 3.12, the transition diagram is classified as non-irreducible, aperiodic (as self-loops exist), and has no unique invariant distribution.

Compute the transition probability

Using the past transition matrix, probability of transition is computed from item 1 to other items using Equation 3.13 .See Table 3.10.

Transition probability between two website links is given in Figure 3.13. Users' *browsing depth* is the sequence or *path count* at the maximum levels (category and subcategory), which a user browses through in order to conclude their purchase or search (Figure 3.14).

Table 3.10: Transition Probability Matrix

Source	Destination	Probability
item 1	item 1	0.3
item 1	item 2	0.4
item 1	item 3	0.2
item 1	item 4	0.1

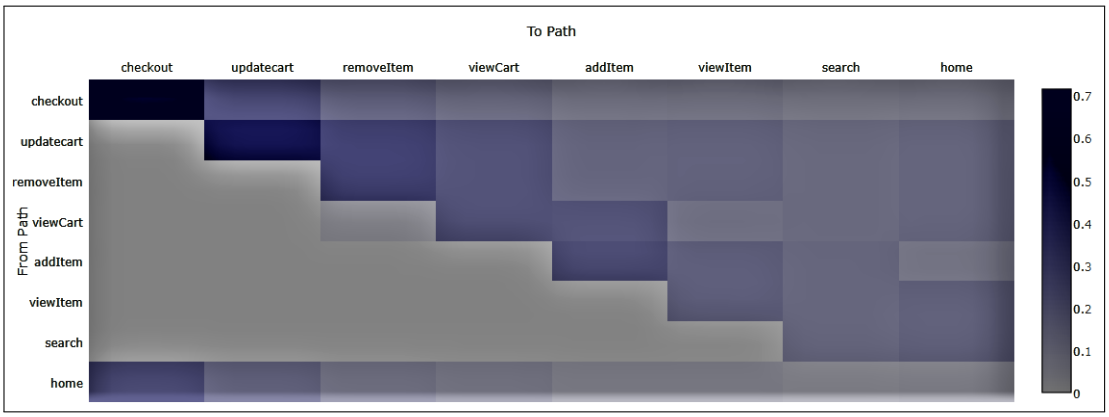


Figure 3.13: Clickstream transition probability between *from path* and *to path*. The transitions are often dominated by transitioning to the same path (such as checkout to checkout or addItem to addItem) and moving back to home.

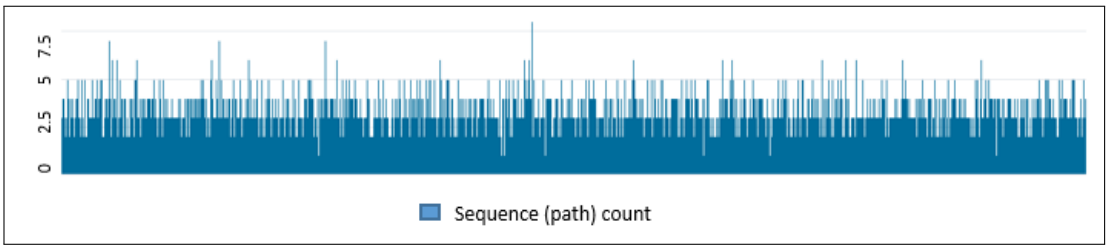


Figure 3.14: Sequence count represents maximum depth (category and subcategory) a user chooses in order to navigate concluding their search. Interpreting the diagram, it is apparent, users frequently browse up to 2.5 levels on average.

3.11 Minimizing Database Latency

The chapter focused on real-time, stream data analytics. Essential for achieving a real-time turnaround time, this section outlines the strategy to achieve milliseconds range response by reducing the database latency. Recall from Figure 3.11, in the Lambda Architecture, post-processing, Cassandra combines and persists the views from both stream and batch. Online Analytical Processing (OLAP) is the model for stream data analysis and exploration, which deals with two *latency constraints*:

- *Data turn around time:* Is the latency between data ingested into the pipeline to the time data is queryable.
- *Query processing latency:* The speed at which results are returned, and reports are generated. OLAP requires interactive query speed in contrast to long-running *batch jobs* taking a few hours to days.

This section first describes the trade-off between highly accurate batch and low latency OLAP. Then it addresses the high volume of read-write operations through the Cassandra datastore and resolves the computational challenge in situations requiring to act faster, such as alerting, processing real-time, or near-real-time.

3.11.1 Trade-off Between Low-Latency and High-Accuracy

The proposed architecture, as depicted in Figure 3.11 and consists of both a low latency real-time component as well as a high-accuracy batch counterpart, working simultaneously on the same dataset. The cost of batch and a real-time algorithm is defined by complexity, i.e. time, space, and resource utilization complexity. We can calculate the competitive ratio of a streaming algorithm against a batch algorithm. A smaller than *one* cost ratio can prove the superiority of the streaming process. However, the competitive ratio, due to time complexity, turns out to be greater than *one* in all scenarios and particularly worse in a certain dataset. This is less surprising because the streaming algorithm processes in mini-batches without having the foresight of the entire dataset.

On the contrary, a real-time algorithm benefits from space and resource utilization complexity due to significantly less volume of data processing in each mini-batch. A real-time algorithm is α competitive if there are positive factors α and γ so that:

$$v_i \leq \alpha v_b + \gamma \tag{3.14}$$

v_i is the cost of the streaming algorithm, and v_b is the cost at the batch setting.

From Equation 3.14, we conclude, an α competitive real-time algorithm has a cost not inferior to α times the optimal batch algorithm (v_b) plus some initial advantages (γ) assigned to an optimized batch algorithm, due to its prior knowledge of the full dataset [160]. Information about the size and schema of the entire dataset in advance is beneficial in terms of using a suitable algorithm and computing infrastructure.

In a classic trade-off between low-latency and high-accuracy, case studies dealing with the real-time dataset select low latency responses over *high accuracy* and *strong consistency*.

The proposed model uses Cassandra DB to support low-latency requirements for the real-time component. Cassandra initiates a *read repair* to update the inconsistent data in a situation with a higher consistency setting. The client read-write processes, therefore, must wait before discrepancies are eliminated. Big data systems that require swift turnaround time at near-real-time can keep the lowest likely value for consistency (which is, *consistency one*) due to the negative effect of consistency levels on general responsiveness. Cassandra allows tunable consistency settings, thus enabling the proposed model to act like a CP (partition tolerant and consistent) or AP (partition tolerant and available) system with regards to the CAP theorem [246].

A database is considered high consistent, if following condition is met:

Definition 3.11.1.

$$r + w > n \tag{3.15}$$

r,w are the consistency levels of read and write, n is the replication factor.

According to definition 3.11.1, for a replication factor of *3*, if read plus write consistency level

is *4* then consistency is considered strong on the fact that read/write operation verifies value from *2* out of *3* replicas.

Definition 3.11.2. *The database is considered eventual consistent or has a weak consistency if the following condition is met:*

$$r + w \leq n \tag{3.16}$$

r, w are the consistency levels of read and write.

n is the replication factor. According to definition 3.11.2, for a replication factor of *3*, if *read plus write* consistency level is *3* or less, then the consistency is considered weak, since the read operation verifies values from *2* out of *3* replicas, and the write operation verifies from *1* out of *3* replicas. This may lead to read-inconsistent data in the case of immediate read requests after a write. However, the database *eventually* reaches consistency with additional delay in read operations [8]. *Eventual consistency* in a distributed systems allow all the replicas in different nodes to *eventually* reach the same state over time, without *locking* the data access. *Eventual consistency* is used to achieve *high availability*, since data is not locked for reading *but* can be *inconsistent* as replicas are still updating over the network.

In the clickstream analysis model, faster read and write is desirable due to the high velocity of data ingestion. Lowering both the write and *read consistency* to *1*, can make the framework responsive at the cost of consistency [8]. However, we can opt for the eventual consistency when higher accuracy is required and follow as per the strategy laid down in Definition 3.11.2.

3.11.2 Data Model

Distributed columnar datastore Cassandra load distributes storage and processing requirements across cluster nodes using a *partitioner* based on a *partition-key*. Data at each partition is further sorted on a *cluster-key*. Near-real-time stream processing applications depend on an efficient data modelling technique for optimal query performance. The following are the constraints for a low

latency response from a distributed system supported by Cassandra:

- A row cannot be *split across two nodes*.
- *Minimize network travel through the data model:*
  We should minimize network load as much as possible. In other words, the fewer the number of nodes the query needs to coordinate with, the better is the overall performance of the data model.

Illustrative Example

The *click_master* table is an entity for the information such as the *session_id, click_sequence*, and the *click_location*. The *click_ticker* table is the second entity storing the *start_time* and *end_time*. Two tables are linked based on the *session_id* column.

click_master(*session_id*, click_sequence, click_location)

click_ticker(*session_id*, start_time, end_time)

Constraints:

- Find duration and click_sequence for a session_id and click_location on a particular day.
- A row cannot be split across two nodes

Data Model: Based on the constraints, we can define the following data model.

```
CREATE TABLE clicks_by_date
(click_location varchar,session_id varchar,
click_sequence varchar,date varchar,
start_time decimal, end_time decimal
PRIMARY KEY ((click_location, date), session_id));
```

Observe, compound primary keys (*click_location, date*) forms the partition key which is ordered by cluster key *session_id*. The *partition key* in this instance is responsible for even distribution of keys across cluster and *co-locates* the keys based on *date* to enable faster *date-based query performance* [94].

3.12 Real-time Processing using Storm

Setting up a Storm Cluster.

A *9 nodes* Ubuntu OS was set up on Microsoft Azure HDInsight Storm cluster. In contrast to Farahabady's approach [77], this work considered a homogeneous Storm cluster setup for Nimbus or Supervisor to make the most out of the default scheduler and load balancer. See Figure 3.15. The cluster's benchmarking results against time is shown in Figure 3.18, 3.19, 3.20.

9 nodes

TYPE	NODE SIZE	CORES	NODES
Nimbus	A3	8	2
Supervisor	D3 v2	16	4
Zookeeper	A3	12	3

Figure 3.15: Microsoft Azure HDInsight Storm cluster with 9 nodes. The cluster consists of 2 Nimbus (master) and 4 Supervisors (worker) and 3 additional nodes for coordinating with Zookeeper servers. The storm cluster operates in speed layer of the Lambda Architecture which associated storage options such as Casandra, MongoDB or HBase.

Creating the Project

- Language: Java Spring
- IDE : Eclipse
- Build tool : Maven

The project structure is shown in Figure 3.16.

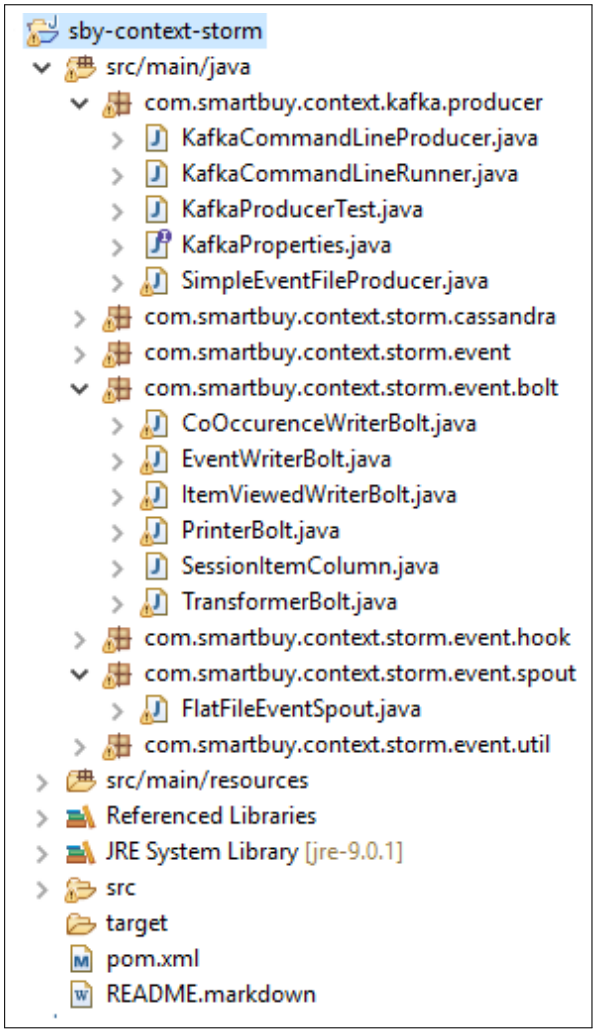


Figure 3.16: Project structure with Java, Maven and Eclipse.

3.12.1 Stream Propagation through Storm Spouts and Bolts

This section provides an overview of real-time event propagation through the series of *Storm Spouts and Bolts* with an objective to achieve milliseconds range response time.

Event Producer. Upstream click events propagate through a Kafka message queue to persist into a file system at real-time.

Storm Spout. Storm *Spouts* consume *Kafka topics* containing clickstream sequences and identify each unique users through a context ID.

Storm Bolts. Series of *bolts* transform each *tuple* of input as follows:

- *Item-viewed* list grouped by the context ID order by timestamp.
- Computes event hierarchy and the overall co-occurrence or *also-viewed* matrix between item pair I_i, RI_i as the sum of each unique session by a user. See Figure 3.17.
- Compute user motifs through n-gram and student-t test.
- Compute customer segmentation through *K-means* and *E-M algorithms*.

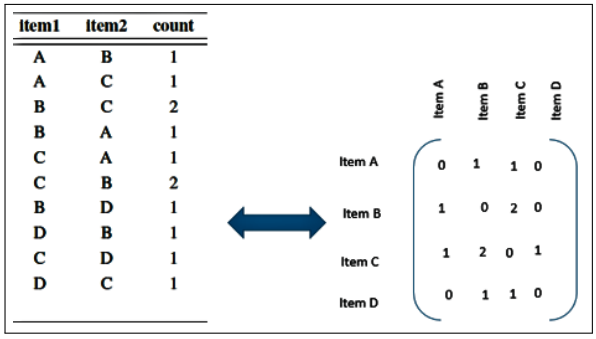


Figure 3.17: Co-Occurrence Matrix between item pair I_i, RI_i for each unique session by a user.

The algorithm is validated in a Storm cluster with the e-commerce data from Amazon and Flipkart (as discussed before). The statistics for throughput, Storm Topology and supervisor summary are shown in Figure 3.18,3.19 and 3.20.

Algorithm 2 Stream Propagation through Storm Spouts and Bolts to Create Co-occurrence Matrix

Input: *S:* Co-occurrence matrix between item pair I_i, RI_i till time t
Past dataset $\mathscr{D}_1$, $\mathscr{D}_2$,...,$\mathscr{D}_t$ untill time t
New dataset $\mathscr{D}_{t+1}$ at time $t+1$
n: Input feature variable
k_t*:* Kafka topic
Message queue: *Linked Blocking Queue*
Kafka Partitioner: *Consistent Random*
Kafka topic k_t, DB server IP d_a,
Output: M: Updated co-occurrence matrix between item pair I_i and RI_i
subscribeKafka(k_t, d_a, queue, partitioner)
for windowed dataset $\mathscr{D}_{t+1}$ **do**
Kafka-instance port $\leftarrow \mathscr{D}_{t+1}$. //Kafka port listening to stream events.
Storm Spout $\leftarrow$ *Flat File Event Spout* //Storm Spouts consumes the file system data.
I_i,RI_i (Storm Bolt) $\leftarrow$ Storm Spout
end for
return I_i,RI_i

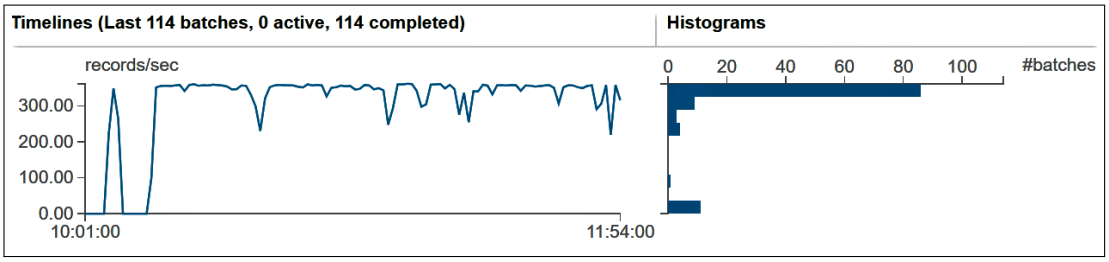
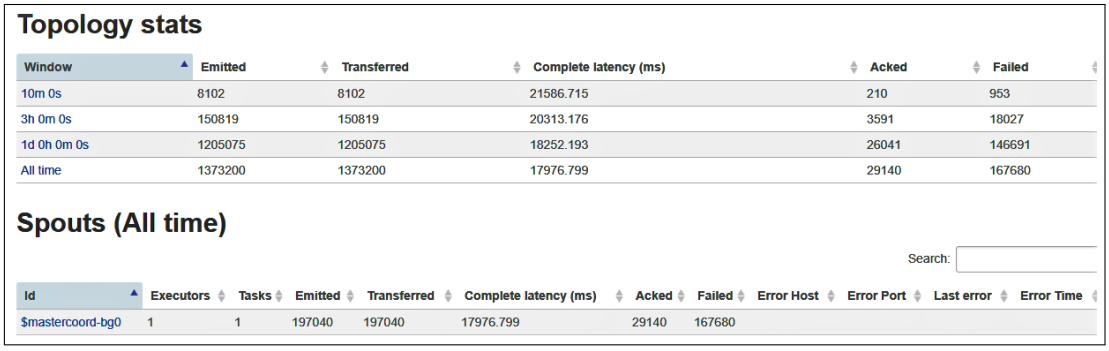


Figure 3.18: Read throughput of the of 114 real-time mini-batches over the period of 1 hour 54 minutes. A read throughput of 300 records/sec is achieved under the streaming setting.



Topology stats

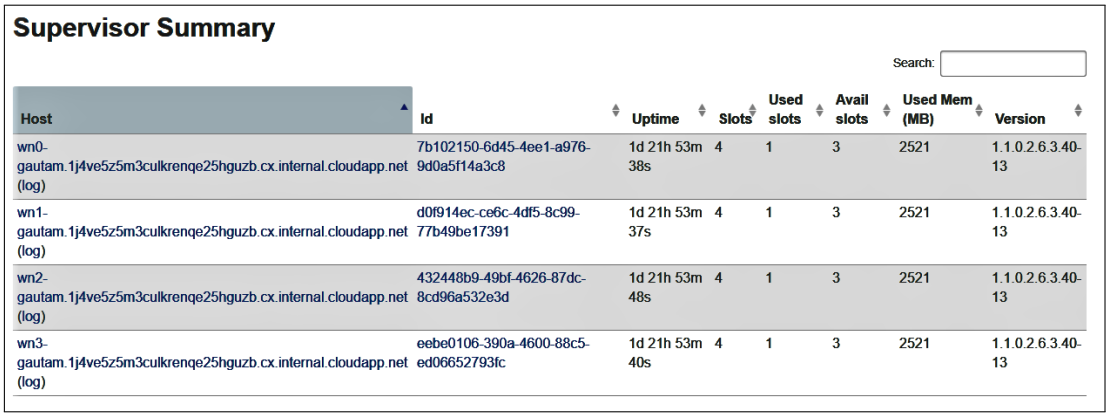
Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	8102	8102	21586.715	210	953
3h 0m 0s	150819	150819	20313.176	3591	18027
1d 0h 0m 0s	1205075	1205075	18252.193	26041	146691
All time	1373200	1373200	17976.799	29140	167680

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
$mastercoord-bg0	1	1	197040	197040	17976.799	29140	167680				

Figure 3.19: Storm topology stats for different time intervals. The statistics shows the efficiency of data processing between spouts and bolts. Failed transactions are retried automatically by the *Storm Nimbus*.



Supervisor Summary

Search:

Host	Id	Uptime	Slots	Used slots	Avail slots	Used Mem (MB)	Version
wn0-gautam.1j4ve5z5m3culkrenqe25hguzb.cx.internal.cloudapp.net (log)	7b102150-6d45-4ee1-a976-9d0a5f14a3c8	1d 21h 53m 38s	4	1	3	2521	1.1.0.2.6.3.40-13
wn1-gautam.1j4ve5z5m3culkrenqe25hguzb.cx.internal.cloudapp.net (log)	d0f914ec-ce6c-4df5-8c99-77b49be17391	1d 21h 53m 37s	4	1	3	2521	1.1.0.2.6.3.40-13
wn2-gautam.1j4ve5z5m3culkrenqe25hguzb.cx.internal.cloudapp.net (log)	432448b9-49bf-4626-87dc-8cd96a532e3d	1d 21h 53m 48s	4	1	3	2521	1.1.0.2.6.3.40-13
wn3-gautam.1j4ve5z5m3culkrenqe25hguzb.cx.internal.cloudapp.net (log)	eebe0106-390a-4600-88c5-ed06652793fc	1d 21h 53m 40s	4	1	3	2521	1.1.0.2.6.3.40-13

Figure 3.20: Storm supervisor summary of 4 worker nodes. The statistics shows the number of worker nodes running within their respective host and corresponding statistics are shown for *id, uptime, slots, used slots and available slots* fields.

3.13 Clickstream Capture and Prediction Model in Microsoft Azure Cloud

Apache Storm cluster is hosted in *Microsoft Azure HDInsight* Cloud environment. Benefiting from the Azure capabilities, the stream capture model is robust in *failover* with no impact in cluster downtime when a worker node fails. The model builds a streaming pipeline with Azure Virtual Network Gateway, which consists of built-in authentication and load balancers for a production-ready deployment. See Figure 3.21 for the Azure deployment model. The performance metrics for the Azure cluster are captured in Figures 3.24, 3.25.

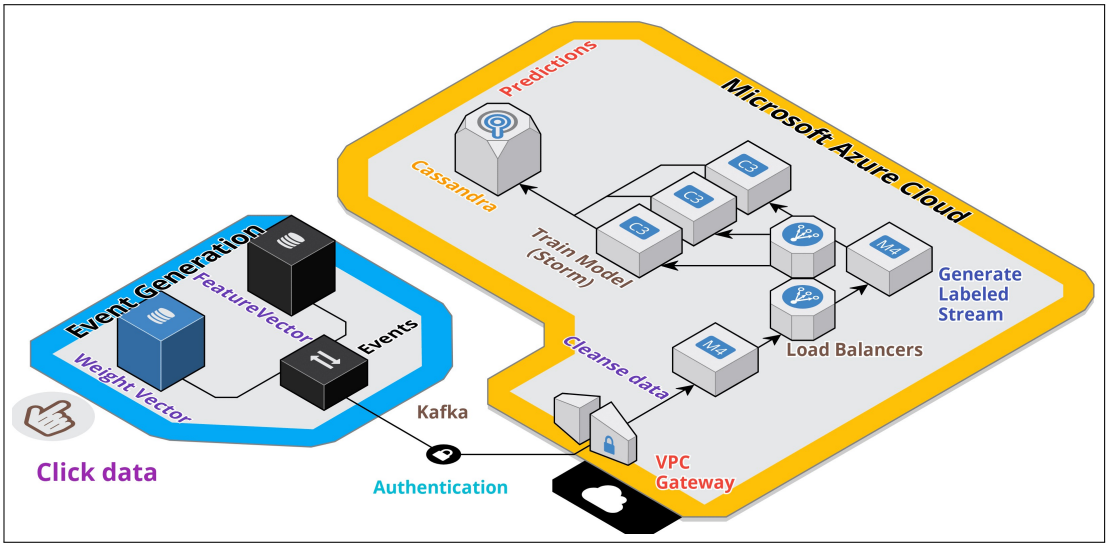


Figure 3.21: Clickstream capture prediction model in the Microsoft Azure cloud. Feature and weight vectors (see Section 3.10.2) are extracted from clickstream in a cluster running the application server. The stream is pushed to the Microsoft Azure cloud through Apache Kafka.

3.13.1 Load Test

This section presents load test results on the DataStax Enterprise (DSE) distribution of Cassandra. 3 million records are used to write first and run a mixed type load (read plus write) for another 3 million records. The server setup is shown in table 3.11. Refer to Figure 3.22 and 3.23 for

test results obtained through Datastax OPSCenter. Figure 3.24, 3.25 show the *average processing latency* in Azure HDInsight storage *unit 1* and *unit 2*.

Table 3.11: Cassandra Setup

Cassandra Vendor	Number of Records	Replication Factor	Number of Nodes
DSE	4 Million writes	3	3

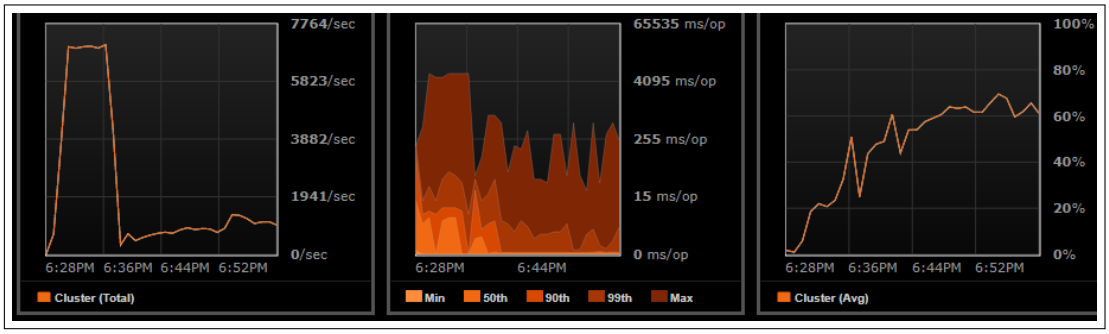


Figure 3.22: Analyzing the number of requests for a given period reveals the underlaying patterns about the read overhead and usage trends. Statistics are shown for *(i) Read Requests, (ii) Write Request Latency, and (iii) OS Disk Utilization*. *(i) Read Requests:* per second read request counts on all coordinating nodes. *(ii) Write Request latency(Percentiles):* 99^{th}, 90^{th} percentiles, minimum, maximum and the median for a client writes. When a node accepts a client read request the period initiates, and terminates while node replies back to the client. Depending on the replication factor and consistency setting, this might include the network delay from the data replicas. *(iii) OS Disk Utilization*: CPU time used by disk I/O. The time unit is in milliseconds.

3.14 Summary

This chapter presented the *first* component of the proposed MAF. The component lets data into the system through the proposed data ingestion methods. The distributed ingestion module is fault-tolerant and suitable for high-velocity big data applications. Real-time ingestion, in-flight processing, and storage approaches were introduced to analyze streams of data with Apache Storm and Cassandra NoSQL datastore. The aim was to build a fault-tolerant distributed ingestion framework

Figure 3.23: Statistics are shown for (i) OS Load, (ii) Heap Used and (iii) TP Flushes Completed. *(i) OS Load:* Operating system load average. Average data for every One minute data are parsed from /proc/loadavg statistics on Linux systems. *(ii) Heap Used:* Average of Java heap space utilized, (iii) *TP Flushes Completed:* Number of memtables flushed to disk since the nodes start.

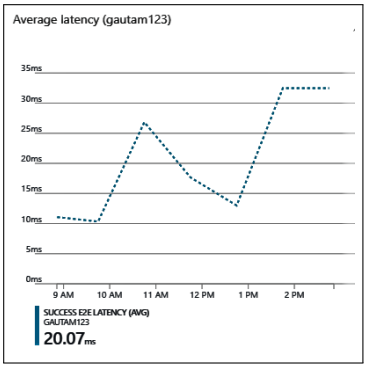


Figure 3.24: Average latency in Azure HDInsight *storage unit 1*, representing the average delay for end-to-end requests sent to a storage operation. The latency time is the total duration required to process within Azure storage unit reading the request, dispatching response, and getting and acknowledgment.

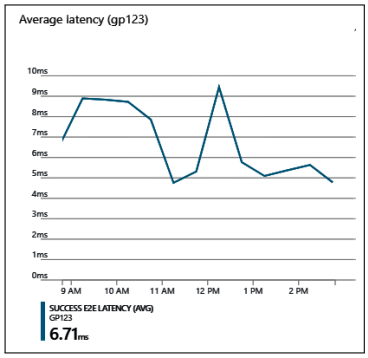


Figure 3.25: Average latency in Azure HDInsight storage unit 2.

on top of the standard big data ingestion tools. The framework provided an efficient solution to horizontal scalability, late-arriving events, and *hdfs small file problem*. This chapter suggested an aggregation mechanism for click events simultaneously generating from multiple sources. Chaining of agents enabled aggregation and propagation of event data in a modular way using Flume *agent-chaining* consisting of three components: source, sink, and channel.

The model was validated using the real-time clickstream analysis for an e-commerce application. The case study presented an aggregation mechanism for click events simultaneously generating from multiple sources. The model analyzed key customer footprint in terms of users' clickstream trail. Click events were captured in real-time using *Kafka*, a distributed big data ingestion framework. *Apache Storm* processes data in real-time at stages (*bolts*) to create a *browsing hierarchy, item co-viewed matrix and frequency of co-occurred items (n-grams)* under a same browsing session. The relative difference of two clickstream distributions on similar click sequences was measured through *Student's t-test*. Closely related users were grouped to create customer segments using a hybrid model of the *K-Means* clustering and the *Expectation-Maximization* algorithm. Initial clusters were created at the batch mode, and the streaming APIs map each user click event to one of the clusters and updates the cluster incrementally at the near-real-time setting. A model was proposed for predicting future click patterns through a *Higher Order Markov Chain*. Processed data was stored into *Cassandra database* to serve real-time responses in the future. This chapter demonstrated several *Cassandra optimization techniques* on a multi-datacenter setup for serving

near real-time responses. Experimental results for a Storm deployment in the Microsoft Azure HDInsight cluster provide important performance metrics. The metrics showed the data for cluster latency with database performance under stress. An approach was shown for building an optimized model for *Storm Topology* serving near real-time responses, as will be discussed in the concluding chapters.

The proposed methods are generic and have potential beyond clickstream analysis with respect to the *Lambda architecture* which combines big data batch and stream processing approaches. In the future, this research will explore wider applications of click pattern behavioural systems and broaden the proposed models to other Human-centered computing (HCC) frameworks.

Chapter 4

Multi-Agent Decision Making Model

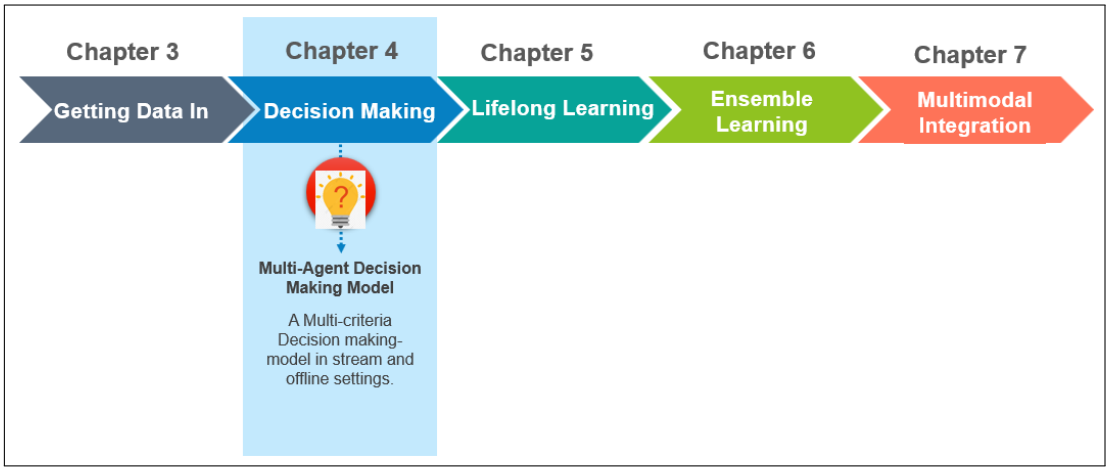


Figure 4.1: An outline of Chapter 4 and end-to-end data flow pipeline in different chapters.

This chapter presents an offline and online (real-time) decision-making system that selects the best-fit methods for each incoming data stream. As shown in Figure 4.1, as part of the proposed MAF, the decision-making component is the *second layer* towards building the data processing pipeline introduced in the thesis. The decision-making component is placed as a gateway to the Multi-agent Lambda Architecture (MALA) presented in the following chapter. The model solves the data *variety* problem of big data by applying different methods for each dataset in an environ-

ment with a *heterogeneous* data pool. A graph-based approach is introduced to imitate real-world problem domains with a set of criteria and problem solvers. *A Multi-criteria decision-making (MCDM) model is proposed* to select a set of problem solvers that best meets the set of criteria. Suppose a system is processing stream data (e.g. Twitter feed) that comes as a stream of JSON records from multiple data sources. The decision system determines which of the available methods to use for a list of requirements (criteria). When multiple criteria, i.e. must meet requirements coexist in a problem domain, their order of importance against the criteria, the mutual influence on each other, and level of indispensability forms a graphic structure. The proposed model considers each vertex of the graph as a *decision criterion* or *benefit of an agent* against the criterion. The connecting edges of the graph denote the mutual influence of multiple agents. This chapter also proposes a fuzzy graph framework to model real-world unpredictability. The model produces benchmarking results for each problem-solver in terms of absolute values to support decision making. The model is implemented through TopBread, Resource Description Framework (RDF), and RDF Data Query Language (RDQL). The key advantage of the proposed model over the existing solutions [40, 88, 122, 291] is that the framework can operate in a dual mode–both as a standalone offline tool and as an online decision-making gateway, it can also be used in high-velocity ingestion scenarios.

In BDA, one of the primary challenges is to process heterogeneous data pools with a single framework. A traditional one-method fits-all approach is not the most efficient for providing the best analytical insight in a scenario where ingested high-velocity, multi-sourced data consists of different formats. Therefore, the need arises for the framework to be agile enough to dynamically apply suitable analytical methods or frameworks among a large number of available options.

The common practice is *try-and-see* in exploring the given dataset. In many cases, incorrect analysis methods are adopted. Consequently, analytics fail in finding important insights, hidden patterns, and deriving value out of the data and in good time. For instance, choosing the right NoSQL database can be based on quality attributes like—availability, durability, consistency, maintainability, recovery time, read-write performance, robustness, reliability, stabilization time, and

scalability. There are four types of NoSQL databases—key-value, column-family, document, and graph database. Each type, in turn, has several dozen implementations (vendors). The proposed framework can facilitate the task to identify the right vendor producing the best results against each database's quality attributes (features) and project criteria (requirements). The model can be used to choose from different machine learning algorithms depending on externally fed criteria and dynamically extracted features of the data.

To overcome the *variety* problem of big data, a novel way is proposed based on a Multi-criteria decision-making model through a graph-based approach to derive appropriate domain knowledge and dynamically fetching suitable problem solvers. The model is validated through solving an online and an offline decision-making problem.

4.1 Summary of Contributions

4.1.1 Graph Based Approach to Represent a Problem Domain

This chapter presents a graph-based approach to imitate a real-world problem domain of a software project with a set of decision criteria and problem solvers. Vertices of the graph denote the set of criteria or benefits of a problem solver agent against the criteria. A list of requirement constraints (availability, durability, consistency, etc.) defines the criteria. If a database is considered as a problem solver agent, each vertex of a graph defines the benefit of the database against each of the quality attributes. For example, assume a relative score, i.e. benefit of using Oracle database with respect to faster read operation is 9, and write operation is 5 for a given dataset. The connecting edges between the vertices are the mutual influences of multiple agents when they coexist in the same problem domain. A fuzzy graph-based framework is proposed to model real-world unpredictability.

The model produces benchmarking results for each problem-solver in terms of absolute values to support decision making by ranking the problem solvers.

The key distinction of the proposed model over the existing ones is that the framework can be

utilized in situations with both offline and online settings. So, decision making is possible *offline* without installing the tools. The model can be used *online* in high-velocity ingestion scenarios through RDQL Jena APIs and produces accurate results in terms of appropriate decision making by complex graph processing.

4.1.2 Big Data Support

The model is designed to solve the *variety* problems in big data. Data is generated in all sizes, formats, and schemas. Hadoop and Spark Input Formats [250] [145] intercept the data at the ingestion layer. However, the fixed rule-based system is insufficient to handle the multi-criteria processing environment. MCDM solves the limitations with a dynamic decision-making scheme. MCDM selects the best-fit methods suitable for data features and structure at every time window of incoming data. When data is non-stationary and patterns changes over time, the selection process evolves as well.

The model is implemented through the Resource Description Framework (RDF), and RDF Query Language (RDQL) is used for query processing. Apache Jena APIs [9] used for querying the graph produce better query latency than SQL in high-velocity big data ingestion scenarios.

4.2 Application Scenarios

Before describing the proposed graph-based decision-making process, this section provides the motivating case studies behind the innovations in terms of online and offline decision making. Case studies are presented in a later section of the chapter.

4.2.1 Scenario 1: Online Decision Making

The online decision-making process works in near-real-time with a pre-configured window interval where all the tools and methods are already running in the cluster. MCDM picks a set of best-fit alternatives using the method described in the next section. Possible case studies include Twitter

stream analysis and large scale fraud detection systems generating data from multiple sources in different formats. The actual implementation has a relatively complex architecture with both batch and real-time processing capabilities. For this purpose, a mixed processing framework is proposed in the thesis called MALA, where real-time and batch components are two collaborative, autonomous agents. First, MALA is briefly described. The proposed MCDM is placed at the entry point of MALA with a decision making task. Both MALA and MCDM are components of the end-to-end Multimodal Analytics Framework (MAF) framework.

Lambda Architecture (LA) [134] [256] [101] is a data processing architecture used in big data for simultaneous processing of real-time and batch data. A real-time layer serves the low latency requests with the most recent data, and a batch layer provides a more comprehensive view of the data through pre-computed results. Based on the fundamentals of LA, a Multi-agent Lambda Architecture (MALA), an ensemble framework, is designed for stream and batch modules. MALA is a three-layered end-to-end data processing framework. The Historical Data Store (HDS) in MALA persists the past dataset into HDFS or in a NoSQL store. The stream processor unit initiates itself with a past dataset in HDS as an offset and incrementally updates the dataset in a streaming window interval. The Knowledge Miner (KM) provides a search and reporting capability. Data cleansing, knowledge aggregation, and data governance are also part of the KM. Refer to Chapter 5 for a detailed architecture overview of MALA.

The proposed MCDM framework is placed at the gateway of MALA. MCDM chooses the best-fit tools and methods at every window length of an ingested data stream. The framework represents the integration service between a data source and a processing engine. A novel Multi-agent, the dynamic selection process, is introduced with the RDF implementation in the data ingestion layer. Dynamic selection finds the best-fit tools and methods available for every window length of an ingested data stream. When data are non-stationary, and patterns change over time, the selection process evolves dynamically. An implementation is shown through the TopBread IDE and using RDFQL query language (See Figure 4.7).

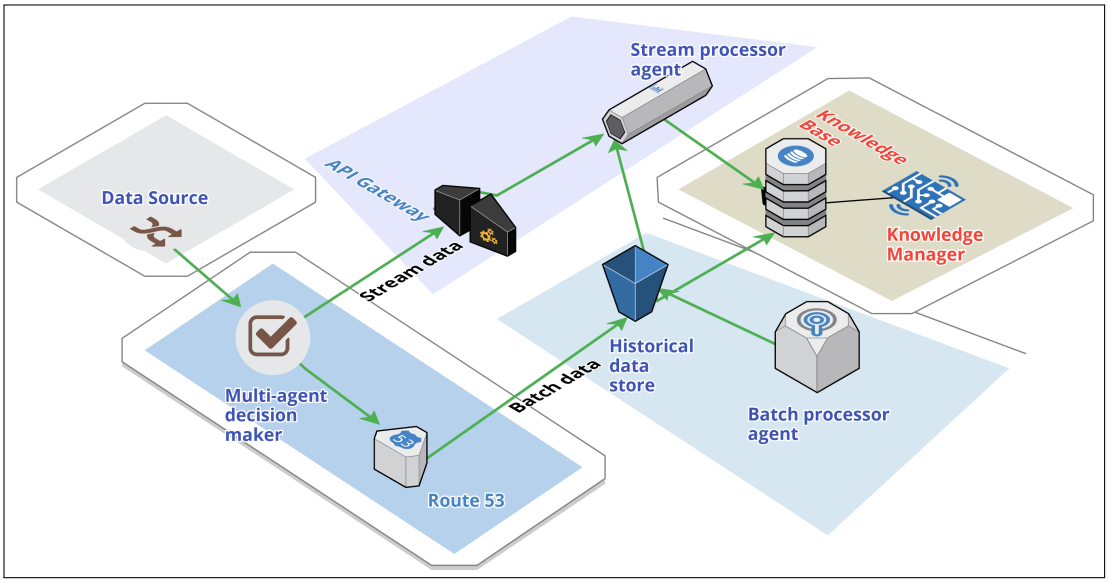


Figure 4.2: Multi-agent Lambda Architecture (MALA). Four layers of MALA comprise of *historical data store, stream processor, Knowledge Base, and Knowledge Miner*. Real-time and batch layers act as autonomous Multi-agent systems in collaboration. MCDM, A Multi-agent decision maker component, is placed into the MALA stack at the gateway of the data pipeline for further processing.

4.2.2 Scenario 2: Offline Decision Making

Another application area is a standalone offline decision-making tool for selecting applications or methods in a multi-criteria problem domain applicable to virtually any IT infrastructure management project or cloud Infrastructure as a service (IaaS) model. The key advantage of an offline strategy is that, unlike online mode, the test applications are, in reality, not required to be installed for benchmarking and decision making. Instead, the proposed mechanism leads the way to a selection process through a completely offline process. A possible implementation for offline decision making may start at the planning phase in a software development project. The MCDM ranks all the available methods so that the best options are selected offline before implementing them.

Section 4.7 contains the test results for both the application scenarios.

4.3 Graph Based Decision Making

In a typical big data ingestion scenario, data propagates through four layers —ingestion, processing, storage, and visualization (See Figure 4.3).

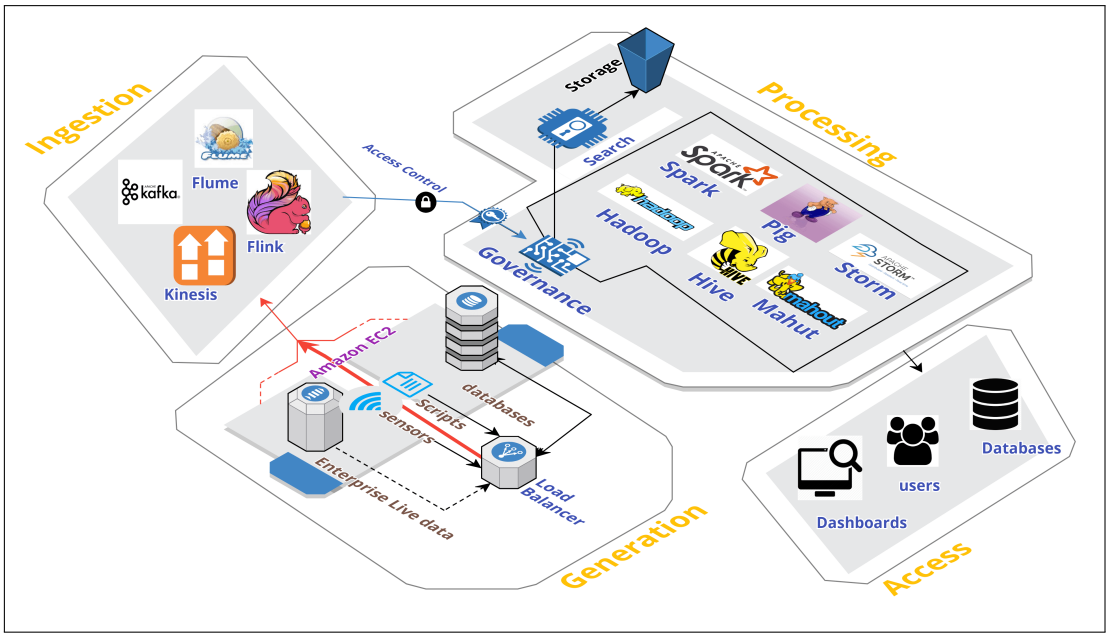


Figure 4.3: Typical four-layered Big Data architecture: ingestion, processing, storage, and visualization. Each layer consists of several alternative tools and methods, and only one needs to be selected for a given dataset. As shown in the Figure, *any one* of the data ingestion tools is selected from a set of alternatives (Kafka, Flume, Flink and Kinesis).

Each layer consists of a set of tools that achieve the same objectives. For instance, in the ingestion layer Kafka, Flume, Kinesis are all used for ingesting the data stream. Similarly, in the processing layer, Storm and Spark are both for stream processing. Hadoop, Hive, and Pig are the distributed processing mechanism based on MapReduce. Based on the requirements and criteria, the purpose is to select *one tool* from each layer and create a data flow pipeline that produces the best results. Since the data formats change over time, data generated from multiple sources and requirements do not remain the same, a smart solution should change methods to process heterogeneous data. In traditional big data systems, the tools, methods, or algorithms in all four layers

in the data pipeline remain unchanged for any data formats or volume. This one-model-fits-all data formats and volume limit overall processing capability, leading to poor insights of underlying patterns. Tight coupling between layers makes the system fragile to change any individual module without changing the overall stack. Popular Business Logic Integration Platforms (BLIP) like Drools and JBPM [197] [217] [146] offer rule-based decision-making engines for multi-criteria event processing. Drools and JBPM manage rule-based decision tables when the rules are managed in a spreadsheet format. The frameworks, therefore, can not consider the graphic structure and discard any mutual influence of rules (criteria) when they coexist and consequently ignore the real-world uncertainties.

In this section, these issues are resolved by introducing a dynamic selection of streaming or batch components in each layer of the big data architecture (Figur 4.3) for each window interval of incoming data. A Multi-agent graph-based decision maker component is placed at the MALA stack as the gateway to the data pipeline for downstream processing. The core idea of the decision-making model is based on the concept of the *collective influence of multiple problem-solver agents* for the given criteria.

Suppose there is a list of alternative problem-solvers P and a set of criteria C. Then, how to choose a problem solver in P, which best satisfies a set of requirements most, is defined as a Multi-Criteria Decision Making (MCDM) problem [142, 158, 173, 290]. This section describes a Multi-agent decision-making model through a graph-based approach to select the best problem solver for a domain.

Suppose, that there are m problem solver agents a_i (i = 1, 2, . . ., m) in A and there are n criteria c_j (j = 1,2, . . . ,n) in C, and all the available dataset forms a matrix M = (α_{ij}) m$\times$ n. Where α_{ij} denotes the criteria value of the problem solver agent a_i against the requirement criteria c_j. Each agent needs to perform against n criteria values. We then rank the problem solver agents that produce the best possible solution. As a decision-making method is needed, an aggregation method is introduced.

We start by plotting a set of agents A = $\{ a_{1,} a_{2,} \ldots, a_n\}$ into a collection of vertices in a graph

G= (A, E), where $E \subseteq [A]^2$ is the set of co-relations among the agents in A.

For a criterion, an agent a_i can take action (i.e. be in an active state) or remain idle, while an agent a_i take an action, denoted by t_i = 1; else, t_i = 0. Then the overall effectiveness of a_i in presence of a_j can be calculated as:

$$b(a_{ij}) = b(a_i) + b(N_i) \tag{4.1}$$

for i=1,2, ...,n

Where

$$b(N_i) = \sum_{j \in N_i} \mu_{ij} b(a_j) \tag{4.2}$$

Where $b(N_i)$ is the set of a_i's neighbors. $b(a_i)$ and $b(a_j)$ are the benefits due to agents a_i and a_j. $\mu_{ij} \in$ [0,1] is the influence coefficient between connected agents a_i and a_j. Influence coefficients denote mutual influence each agent has on one another connected agent when they co-exist in a global problem set. The Equation 4.3 defines the initial value of the Influence coefficient:

$$\mu_{ij} = \frac{b(a_i) + b(a_j)}{k} \tag{4.3}$$

where $b(a_i)$ and $b(a_j)$ are the benefits of agents a_i and a_j. k is a normalization constant. The value of k is any integer between the average to the largest of all the vertices in the graph.

In Equation 4.2, if μ_{ij} = 0, for j$\in N_i$ (i = 1,2,...,n), then b(a_{ij}) = b(a_i) by the Equation 4.1. It implies that, an agent's neighbors have no influence on the overall benefit for an agent. The agent just acts alone and the connected graph structure is of no use. In principle, the more connected an agent (or more neighbors it has), the more important the agent is. Hence, the degree of connectivity of a vertex is a conclusive factor to compute the overall importance of an agent.

4.3.1 Adjusting the Influence Coefficient Values

Initial coefficient values between two nodes are computed using Equation 4.3. However, if a node is also a common neighbor of other nodes, then the influence coefficient is further updated. For instance, in Figure 4.4, if $b(a_1)$ is 10 and $b(a_2)$ is 5 then we calculate the influence coefficient $\mu_{12} = \frac{(10+5)}{10} = 1.5$. Suppose, node a_1 gets *two* new neighbors as a_2 and a_3 (Figure 4.5), but a_2 and

Figure 4.4: Initial coefficient values between two nodes.

a_3 themselves are neighbors to each other. We update the coefficient by increasing the coefficient

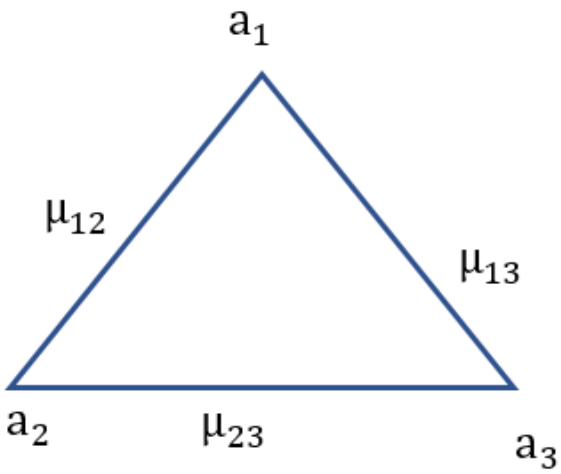


Figure 4.5: Coefficients are updated with addtion of new nodes a_2 and a_3

values to the number of common neighbors they share. Hence, the equation is updated as:

$$\mu_{ij} = \frac{b(a_i) + b(a_j)}{k} \times (c_{a_{ij}} + 1) \tag{4.4}$$

where $c_{a_{ij}}$ is the number of common neighbors between node a_i and node a_j. The equation implies the mutual influence agent vertices a_i and a_j have on each other when they both coexist in a problem

domain.

4.3.2 Considering Agent Priority

In a real-world scenario, all the agents are not at the same priority level for a given criterion. For instance, in a banking transaction processing system where consistency is essential when storing the data in a database. In such a scenario, the priority of an RDBMS may be higher than the priority of a NoSQL. Similarly, under project budgetary constraints in a production deployment, open-sourced software may be put on a higher priority compared to licensed software.

In terms of agent priority, according to the role, the proposed method assumes that the more connected a vertex is, the more important the agent becomes and the more priority it has on the overall benefit of a graph in terms of a numerical value. The degree or the weight of each agent vertex a_i is normalized as

$$\omega_i = \frac{d(a_i)}{max_i(d(a_i))} \tag{4.5}$$

where $d(a_i)$ is the degree of each node in terms of the number of edges connected to a vertex, and $max_i(d(a_i))$ is the maximum degree of a node in the graph.

Then overall benefit of the plan as represented by the graphic structure is calculated as:

$$b(a_i) = \sum_{i=1}^{n} \omega_i \sum_{j=1}^{N_i} b(a_{ij}) \tag{4.6}$$

where ω_i is the normalized weight of the i$^{\text{th}}$ vertex as defined in Equaltion 4.5. $b(a_{ij})$ is the benefit of the i^{th} vertex with the presence of the j^{th} vertex. The i^{th} and j^{th} vertices are the neighbors as defined in Equation 4.1 and 4.2.

Refer Section 4.7.1 where the proposed equation is applied to evaluating a set of problem solvers.

4.4 Decision Making through Fuzzy Graph

The decision-making process presented in the previous section can also be implemented through a fuzzy graph-based approach for greater accuracy. In fuzzy graphs, values of vertices and edges can be variable instead of a fixed value. Therefore, fuzzy graphs are more realistic in terms of capturing uncertainties in the actual scenarios.

First, a few basic concepts and terminologies are introduced with regards to fuzzy graph theories, which are used throughout this section. Definitions of *fuzzy vertex* and *fuzzy edge*, *path*, *diameter* and *strength* are given. Specifically, in this chapter, *strength* of the path in a fuzzy graph is considered as the mutual influence of connecting vertices instead of every edge to recompense the real-world unpredictability.

Definition 4.4.1. *A fuzzy graph is defined by G = (V, α, β). The triple consists of a nonempty set V where α:V $\rightarrow$ [0, 1] and $\beta : \varepsilon \rightarrow$ [0, 1] such that for all $v_1, v_2 \in$ V, $\beta(v_1v_2) \leq \alpha(v_1) \wedge \alpha(v_2)$. The instances of V are called as vertices of the graph. α is the fuzzy vertex set of G, and β is the fuzzy edge set of G. The symbol $\wedge$ is the minimum of two vertex values assigned as a fuzzy edge (β).*

Definition 4.4.2. *A path in a fuzzy graph G = (V, α, β) is a set of vertices $v_0, v_1, ..., v_n$ so that $\beta(v_{i-1}v_i) > 0$ i=1,2,...,n. n is the length and the longest path between v_{i-1} and v_i is called the diameter.*

Definition 4.4.3. *A strength of the path in a fuzzy graph G = (V, α, β) is defined by weight of the lowest edge expressed as $\wedge_{i=1}^{n}\beta(v_{i-1}v_i)$.*

Let A' be an uncertain (variable value) fuzzy set defined by a fuzzy set a_i', where a_i' is the set of fuzzy vertex α_i where i = 1.2,...,n and $0 \leq \alpha \leq 1$. The fuzzy relationship between agents a_i and a_j is described as β where $0 \leq \beta_{ij} \leq \alpha_i \wedge \alpha_j$; i, j = 1, 2,..., n. The agents and their fuzzy relations is defined in the Fuzzy graph. Each agent represents a problem solver. Suppose there are m problem solver agents a_i (i=1,2,...,m) in A and there are n criteria c_j (j=1,2,...,n) in C. We rank the problem solver agents producing the best possible solution using the fuzzy based aggregation and decision

making method. Considering the graphic structure of the connected agents, the overall benefit of an agent is calculated by the benefit of the agent itself plus the collective influence of all the connected vertices in the graph as defined by Equation 4.7.

$$\widetilde{b}(a_{ij}) = \alpha\widetilde{b}(a_i) + \widetilde{b}(N_i) \tag{4.7}$$

for *i=1,2, ...,n*, where

$$\widetilde{b}(N_i) = \sum_{j \in N_i} \beta_{ij}\mu_{ij}b(a_j)) \tag{4.8}$$

where $b(N_i)$ is the set of the a_i's neighbors.

$\mu_{ij} \in [0,1]$ is the influence coefficient between connected agents a_i and a_j. The fuzzy graph can represent real-world situations (an illustrative example follows in the next section) where an agent's mutual influence (represented by edges of the graph) is not known exactly and in a natural sense, the relationship is *fuzzy*. Instead of using the adjacent path between vertex v_{i-1} and v_i, the weight of the lowest edge between vertex v_{i-1} and v_i is used as the *strength* of the graph according to the Definition 4.4.3. The *strength* of the graph represents a fairly consistent value compared to an individual edge. Then the overall benefit (in terms of a numerical score) of the scheme in presence of agents a_i and a_j is

$$\widetilde{b}(a_i) = \sum_{i=1}^{n} \gamma_{ij} \sum_{j=1}^{N_i} \widetilde{b}(a_{ij}) \tag{4.9}$$

Where γ_{ij} is the strength of the fuzzy graph as defined in Definition 4.4.3. The end-to-end process is given by Algorithm 3.

lines 3-4: Benefit of an agent $(b(a_{ij}))$ is calculated by the benefit of the agent itself $(b(a_i))$ plus the benefit of neighbors $b(N_i)$.

lines 5-6: Compute benefit of neighbors $b(N_i)$.

lines 7-8: Compute the mutual coefficient between two connected agents a_i and a_j.

lines 9-11: Overall benefit of the plan (as defined by the criteria) is computed.

Algorithm 3 Multi-Criteria Decision Making Model

Input: a_i: Problem solver agents
c_j: Requirement criteria
d_b: historical batch datastore
$d_s = \sum_{i=1}^{n} d_i$: dataset as a collection of streaming records since the last window
Output: a_i: Ordered set of problem solver agents

Initialize stream engine with batch datastore
for each d_s **do**
  $b(a_{ij}) = b(a_i) + b(N_i)$
  $\widetilde{b}(a_{ij}) = \alpha\widetilde{b}(a_i) + \widetilde{b}(N_i)$ (for fuzzy graph).
  $b(N_i) = \sum_{j \in N_i} \mu_{ij} b(a_j)$ is the set of the a_i's neighbors.
  $\widetilde{b}(N_i) = \sum_{j \in N_i} \beta_{ij}\mu_{ij} b(a_j)$ (for fuzzy graph).
  Coefficient $\mu_{ij} = \frac{b(a_i)+b(a_j)}{k} \in [0,1]$ (initial value)
  $\mu_{ij} = \frac{b(a_i)+b(a_j)}{k} \times (c_{a_{ij}} + 1)$
  Normalized weight $\omega_i = \frac{d(a_i)}{max_i(d(a_i))}$
  $b(a_i) = \sum_{i=1}^{n} \omega_i \sum_{j=1}^{N_i} b(a_{ij})$ (overall benefit)
  $\widetilde{b}(a_i) = \sum_{i=1}^{n} \gamma_{ij} \sum_{j=1}^{N_i} \widetilde{b}(a_{ij})$ (for fuzzy graph)
end for
return *sort*$(b(a_i))$

4.5 Illustrative example

Suppose a software project makes a selection of applications from the several available options, based upon the 3 main criteria prioritized by the actual requirements laid out in the case study: consistency (a_1), scalability (a_4), high read throughout (a_3) and availability of an open-source license (a_2). The project requires a high level of consistency as it deals with critical financial data. The project starts with a few terabytes of input data to be analyzed and may not require scaling up frequently (low scalability). The analytics jobs are run as batches at a 6 hours interval (medium throughout). Analyzing the requirements laid out, the requirement criteria can be prioritized as a_1> $(a_2, a_3) > a_4$. On the other hand, the project can choose an application that must meet the criterion a_1. Criteria a_3 and a_4 are not mandatory but good to have; and criterion a_2 is the least preferred among all. The relationship among the criteria a_j (j=1,2,3,4) can be expressed by a complete graph G=(A, E) as shown in Figure 4.6: In general, the more connected a node is, the more important

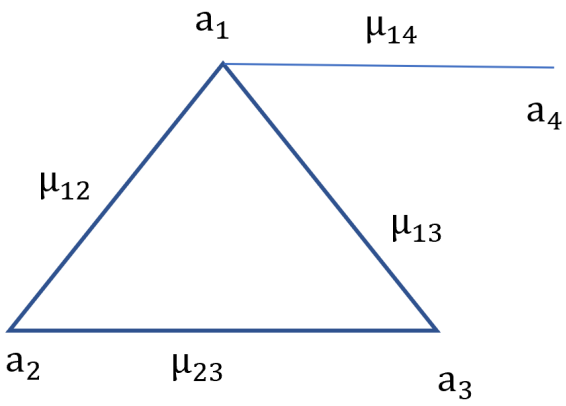


Figure 4.6: Graphical view of coexistence of multiple criteria (or the benefit of an agent against the criteria). Vertices represent a criterion, edges are mutual coefficient between two criteria when both coexist in a problem domain.

it is considered to be in comparison with other nodes. As the project puts the highest priority on consistency, represented by node a_1, it has the highest number of edges (three) connected to it. Now we apply the MCDM to calculate the overall benefit score of an application.

Step 1: Calculate the weight for each vertex.

From the Figure 4.6, calculate the degree for each vertex: $d(a_1) = 3$, $d(a_2) = 2$, $d(a_3) = 2$, $d(a_4) = 1$ After normalizing the degree of each vertex by Equation 4.5:

$\overline{d}(a_1) = \omega_1 = 1$

$\overline{d}(a_2) = \omega_2 = \frac{2}{3}$

$\overline{d}((a_3) = \omega_3 = \frac{2}{3}$

$\overline{d}(a_4) = \omega_4 = \frac{1}{3}$

where ω_i is the weight for each vertex.

Step 2: Determine the benefit of a problem solver agent against a set of criteria.

Four criteria of consistency (a_1), scalability(a_4), high read throughout (a_3) and availability of an open source license (a_2) are ordered as: $a_1 > a_2, a_3 > a_4$. The benefit for each application represented by degree of the graph vertices as follows: $b(a_1) = 3$, $b(a_2) = 2$, $b(a_3) = 2$, $b(a_4) = 1$.

Where $b(a_i)$ is the benefit of an application independently which is determined based on the level of connectivity or the degree of each vertex.

Step 3a: Calculate the influence coefficients.

Influence coefficients are calculated using Equation 4.3 for the graph 4.6 as follows: $\mu_{12} = 0.8$, $\mu_{13} = 1$, $\mu_{23} = 0.6$, $\mu_{14} = 0.5$

Step 3b: Calculate the influence coefficients for the *fuzzy graphs*

According to *Definition 4.4.3*, influence coefficient of a Fuzzy graph is the weight of the weakest edge. So, values of the influence coefficients are: $\mu_{12} = \mu_{13} = \mu_{23} = 0.6$, $\mu_{14} = 0.5$

Step 4: Calculate the overall benefit by Equation 4.6

$b = \omega_1 \times (b(a_1) + \mu_{12} \times b(a_2) + \mu_{13} \times b(a_3)) +$

$\omega_2 \times (b(a_2) + \mu_{12} \times b(a_1) + \mu_{13} \times b(a_3)) +$

$\omega_3 \times (b(a_1) + \mu_{12} \times b(a_2) + \mu_{13} \times b(a_3)) +$

$\omega_4 \times (b(a_4) + \mu_{14} \times b(a_1))$

b = 13.7

b = 13.13 (For a fuzzy graph).

Step 5: Repeat the process for the rest of the problem solver agents.

Repeat *steps 1 to 4* to calculate the overall benefit for each of the test applications and select the *top n* applications by comparing the benefit score of each.

4.6 Implementing Multi-Criteria Decision-Making Model (MCDM)

The MCDM model discussed in the preceding section is implemented through a Resource Description Framework (RDF). RDF represents semantic web modeling by extending the linking structure of the web to use URIs defining a relationship between elements as well as the two ends of the link, referred to as a triple.

On every mini-batch of streaming data, the feature of the data is extracted. Extracted features from input data such as (i) data formats: CSV, JSON or XML, (ii) the shape of the regression function: linear or arbitrary, along with (iii) externally fed features (by users) of the input data such as loss function, generative or discriminative function, categorical or numerical feature, altogether are used to draw the graph as the query parameters of the RDF. The results of the queries select appropriate methods and establish a link between multiple methods. The dynamic selection of resources from generation to consumption takes place for every window of incoming stream data. See section 4.7.2 for further details with a comprehensive implementation.

For example, consider processing incoming Twitter streams in the format of JSON or CSV. Data generated from multiple sources containing textual data, i.e. categorical features, extracted from an input stream. Requirement criteria are externally fed, such as Random Decision Forest as a machine learning method. Twitter streams are processed in real-time with a highly available cluster setup (considered as a criterion). All the data features and requirement criteria form a graphic structure. If there are n available alternative methods: $m_1, m_2, ..., m_n$, the proposed MCDM ranks the methods to select the best options.

TopBraid Composer Maestro Edition (IDE) models the RDF [269] [171]. TopBread Composer is an easy to build powerful semantic web, linked data, enterprise-grade application. Figure 4.7 represents a screenshot of the RDF modeling. Note that MCDM has an infrastructure model that is extended by ingestion, processing, storage, and a visualization model. Extending further, storage

is derived by column store, document store, and relational store. The column store has DynamoDB as one of the instances.

Programmatically, RDF Data Query Language (RDQL) processes queries. RDQL uses SQL like syntax to fetch data from the RDF store. Jena, a full-featured Java API, is used for the RDF. A Java layer implements the MCDM model discussed in the preceding section. The following are the advantages of the MCDM model:

- *Loose coupling* between layers from data generation to consumption enables enterprises to change the underlying elements without changing the entire stack. MCDM picks different methods based on the data and criteria by comparing alternative methods that deliver the same functionalities. Therefore, loose coupling is a prerequisite for the MCDM.

- *Dynamic selection* of components for each window length of data optimizes ingestion, processing, and storage capabilities by selecting best-fit methods in hand, resulting in greater analytical insights.

- *RDF representation* helps join data from two disparate vocabularies easily without having to negotiate the structural difference between the two [135] [139]. The RDF is modelled in TopBraid Composer (IDE) (Details in section 4.7).

4.7 Experimental Results and Analysis

In this section, MCDM online and offline decision-making model results are presented. At the offline setting, MCDM selects the most suitable database (as a problem solver) that most satisfies a set of required criteria. In the online setting, MCDM selects one of the installed applications in the current scenario, a machine learning algorithm, for each streaming window of the dataset. Details of the test procedure are as follows. The primary differentiator between online and offline mode is, the online mode is a real-time decision-making process and components are installed and running in the setup.

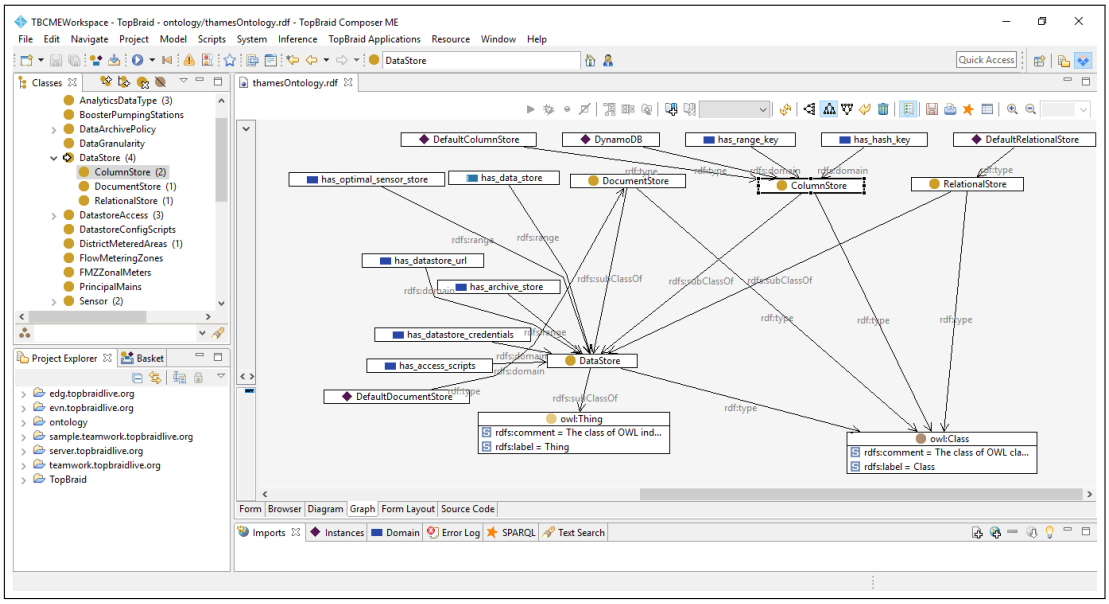


Figure 4.7: TopBraid Composer Maestro Edition (IDE) is used to model the RDF. The Figure represents a screenshot of the RDF modelling. Note that MCDM has an infrastructure model that is extended or derived by an ingestion model, processing model, storage model, and visualization model. Further, storage is derived by column store, document store, and relational store. The idea is to expand the graph from a root as a *generic category* to leaves as *specific instances*, and draw the edges as mutual influences of multiple leaves. RDQL queries the graph to find an overall score (explained in section 4.7.1).

4.7.1 Offline Decision Making

The offline model uses *three* NoSQL databases as test candidates for evaluations: Cassandra, HBase, and MongoDB. Databases are evaluated with respect to their overall benefit for the set of tasks using the proposed algorithm. Offline evolution does not require test candidates installed. However, while *validating* the offline results requires databases installed in a cluster setup.

The offline process does not require an application installed in the cluster, providing the way to compare and select the best offering *offline*, against a set of criteria. For evaluation, an analytics project is chosen, which does a significant number of database operations with an almost equal amount of read and write operations, i.e. balanced load with 50% read and 50% write operations. We consider each specification as a criterion a_1. The project also required a highly available system (a_2) with substantial high throughput (a_3). At the same time, it processes a large amount of historical batch data and does not require the database to be very much consistent (a_4). We order the criteria (a_i) in terms of overall importance:
Balanced read-write operation $(a_1) >$ highly available (a_2), high throughout $(a_3) >$ consistency (a_4).
A graphical view of the project dependencies is represented in Figure 4.6. The vertices a_i denote a criterion, edges μ_{ij} represent mutual coefficient between two criteria when both agents co-exist in a problem domain. Note, vertices can be represented as *criteria* a_i or as *benefit* $b(a_i)$ of an agent a_i against the criteria c_i. Table 4.1 lists benefits for a set of NoSQL databases against each of the criteria independently.

Note that databases are the test candidates in the example; multiple alternative databases *do not coexist* in the same domain but *multiple criteria can be there* with mutual inference to each other. The value of each vertex is the score of each database when installing alone.

Applying the Equation 4.4 we calculate the coefficients (table 4.2). We can calculate the weight (i.e. normalized degree of each vertex) by Equation 4.5. See Table 4.3. Then, we calculate the overall benefit of each test application by Equation 4.6. See Table 4.4.

The result shows that the *Cassandra* has a clear advantage in terms of the laid down require-

Table 4.1: Benefit of databases against the set of criteria

NoSQL DB	$b(a_1)$	$b(a_2)$	$b(a_3)$	$b(a_4)$
MongoDB	1	1	1	3
Cassandra	3	3	3	1
HBase	2	1	2	3

NoSQL databases listed are the problem solver agents and the test candidates. $b(a_i)$ represents benefit of a problem solver agent against a criterion a_i, considering the graphical structure as shown in Figure 4.6. Note that, individual benefit $b(a_i)$ shown here are predetermined based on the known strength of a database against a criterion a_i. MCDM aggregates them to compute an overall benefit score considering the mutual influence of multiple criteria.

Table 4.2: Coefficient metrics between two vertices

NoSQL DB	μ_{12}	μ_{13}	μ_{14}	μ_{23}
MongoDB	0.4	0.4	0.4	0.4
Cassandra	1.2	1.2	0.4	1.2
HBase	0.6	0.8	0.5	0.6

μ_{ij} represents mutual influence coefficient of agent i and j, when they both co-exists in a specific problem domain. Mutual coefficients are computed through Equation 4.4 for the graphic structure shown in Figure 4.6. Coefficient values are used to obtain an overall benefit score.

Table 4.3: Normalized vertex weights

$\omega 1$	$\omega 2$	ω_3	ω_4
1	0.66	0.66	0.33

Table 4.4: Overall Score

MongoDB	Cassandra	HBase
6.5	25.6	12.6

Overall score relative to the requirement constraints.

ment criteria. Refer to the following section to compare offline test results with results from actual cluster installations (online tests).

Online Evaluation

Once the absolute score is computed for each of the NoSQL databases against the requirement criteria, we compare and validate results with online performance evaluation, i.e. the actual independent cluster installation of the databases. The project starts with *two* nodes cluster and scales up to 4 nodes to 8 nodes. The test setup used the Amazon Cloud Platform (AWS) Compute Engine. Instances used Centos 7 operating system; machine type was 4 vCUPs, 15 GB memory. Cassandra version used was 3.11.0, MongoDB version was 3.4, and the HBase version was 1.2.6.

1. *Workload Selection:* Data volume is representative of a big dataset exceeding the RAM capacity of each node. Workload was *mixed balanced type*, containing the read/write combinations: 50% read, 50% write/update.

2. *Throughput by workload:* Throughput (read/write/update per second) is measured, which is graphed vertically. The number of nodes used for the workload is displayed horizontally.

Remarks

We can now compare results received from the online method (Figure 4.8) with the absolute score from offline MCDM output provided in Table 4.4. Cassandra proves to be the most suitable option for the given constraints, followed by HBase and MongoDB, showing the effectiveness of the proposed offline procedure.

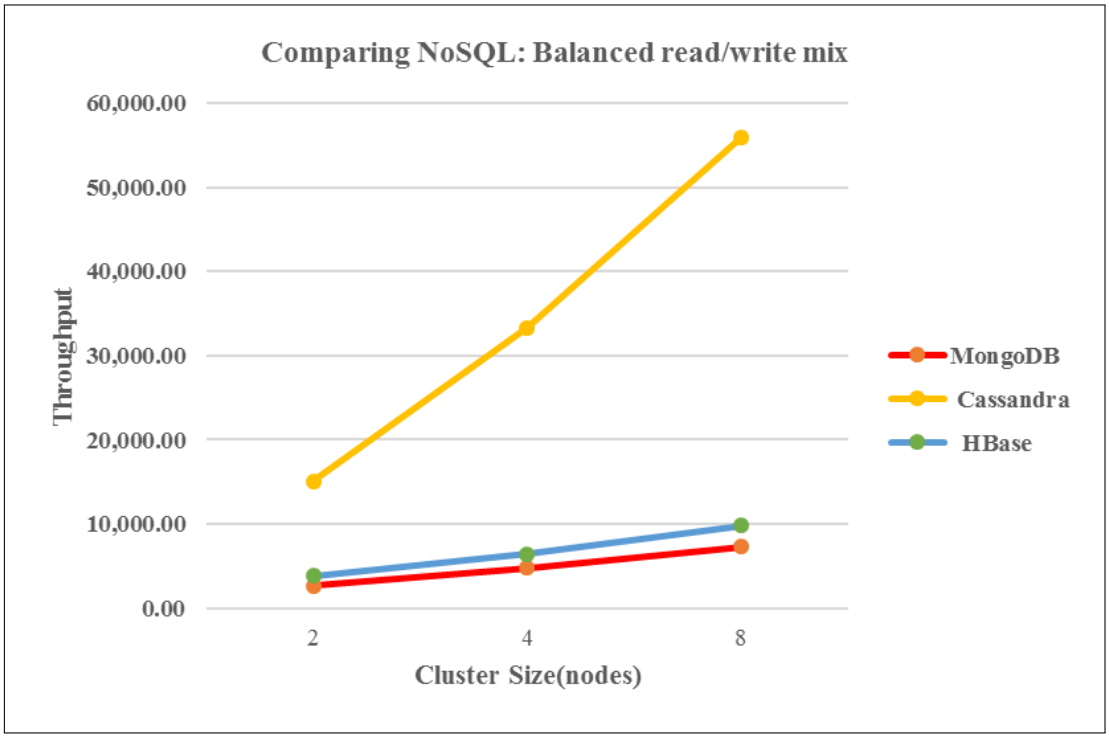


Figure 4.8: Throughput over workload of *mixed balanced* type, containing the read/write combinations: 50% read, 50% write/update. The results are compared against offline MCDM results provided inTable 4.4.

4.7.2 Online Decision Making

Online decision making takes place on each sliding window length of data, at near-real-time but not on each record. Figure 4.9 shows an end-to-end flow of a real-time data processing architecture in a combination of a batch. The implementation shown here uses Apache Storm as a stream processor and Cassandra for data storage and is a component of MALA (Figure 4.2).

MCDM, an online decision maker component, is placed into the MALA big data stack at the gateway of the data pipeline for further processing. In the online setting, MCDM selects best-fit algorithms at every streaming window of ingested data.

Predictive analytics tasks are used to train on the dataset using different machine learning algorithms in *two* different settings: (i) each algorithm runs individually on a standalone installation,

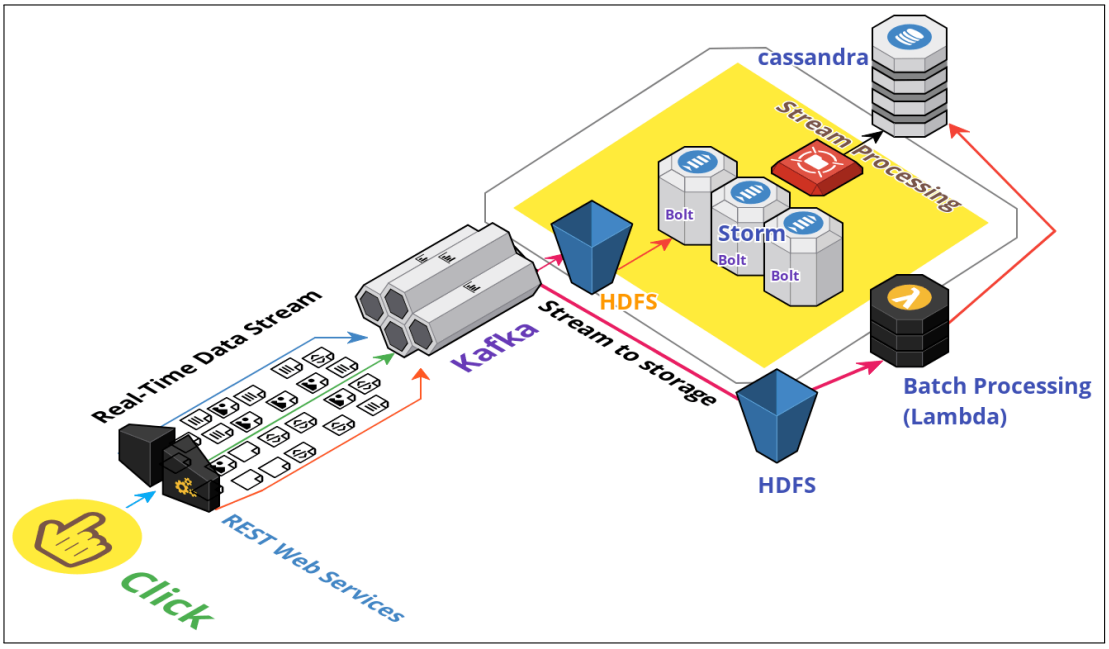


Figure 4.9: The architecture shows the end-to-end flow of real-time data processing in a combination of a batch. Lambda Architecture blends a high throughput Hadoop batch framework with low latency real-time frameworks over a large distributed setup. Note, while data is processed in real-time as Storm Spouts and Bolts, concurrent processing with the batch setup is carried out through the historical dataset out of HDFS. Cassandra combines the views from the stream and batch.

and (ii) in a computing environment where all of the algorithms are available for execution along with the MCDM decision maker component selecting one method at each instance. The experiment runs the algorithms in two different settings against the same data for comparisons in terms of R^2, *RMSE, precision, recall, accuracy* and *F1 Score*. The dataset for this research uses the National Cancer Institute's Genomic Data Commons (GDC) unified data repository [10] for the cancer research community *in support of personalized medicine*. The repository contains over 32000 patient cases, including clinical data, treatment data, biopsy results, gene expression data, as well as a host of other pathological information.

MCDM chooses a single method for a window interval based on the *externally fed criteria* and *dynamically extracted features* from the incoming data. Externally user fed criteria are loss function, generative or discriminative function, categorical, or numerical feature. Several criteria

are inferred from the input stream, such as (i) the input data formats as CSV, JSON, or XML, (ii) regression function Shape like linear or arbitrary. We can compare MCDM with *Tree Regressor, Kernel Ridge, Elastic Net, Ridge, Linear Regression, Lasso models* for numeric fields, and *Gaussian NB, Bernoulie NB, Decision Tree Classifier, SVM and Logistic Regression* for categorical fields.

For example, the online mode of MCDM can be used as a real-time decision-making aid for prescribing personalized medicine. The system receives incoming data in the format of JSON or CSV generated from multiple sources and contains numerical data extracted from the input stream. The system is *externally fed with criteria* such as the *Decision Tree Classifier* as a machine learning method to train the system. The system should be capable of processing high-velocity data in real-time (criteria). All of the data features and requirement criteria form a graphic structure. If there are n available alternative methods as $m_1, m_2, ..., m_n$, proposed MCDM ranks the methods to select the best options. As the data characteristic or requirement criteria change over time, the real-time decision-making component selects different components and keeps updating the training model.

For numerical fields, algorithms are compared in terms of R^2 statistics and Root Mean Square Error (RMSE). In general, the higher the R^2 and the lower RMSE signifies the model fits the data better. Model prediction accuracy for numerical and categorical fields is presented in Table 4.5 and 4.6. In categorical forecasting, algorithms are compared in terms of *Precision, Recall, Accuracy,* and *F1 Score*. A good model is supposed to exhibit higher values for all comparison parameters. The training and test dataset was split at 50% for both experiments.

Categorical prediction accuracy for MCDM is 20% greater than SVM, 41% greater than Bernoulie NB, and more than 43% greater than all other algorithms. In terms of numerical prediction RMSE, the proposed model is 25% better than the Elastic Net, 64% better than Decision Tree Regressor, and more than 139% better than the rest of the algorithms. Therefore, MCDM improves classification and regression accuracy significantly for multiple scales and parameters compared with test candidates.

A *Confusion matrix* is also produced to evaluate the quality of the MCDM categorical classi-

fier's output. The confusion matrix is a $n \times n$ matrix where n is the number of possible outcomes. Values across the rows represent real values, while predicted values are placed along with columns. The entry at row i and column j represent a count of the number of times an instance with true category i was predicted as a category j. So, along with the diagonal cells, the counts for the correct predictions are placed. Counts for incorrect predictions are placed in the rest of the cells (Figure 4.10).

Table 4.5: Model Prediction Accuracy for Numeric Fields

Algorithm	R^2 **Statistics**	**RMSE**
Decision Tree Regressor	-0.9600	1149.84
Kernel Ridge	-4.4322	1678.87
Elastic Net	-0.2219	875.24
Ridge	-12.8802	3032.13
Linear Regression	-8.1000	2213.07
Lasso	-13.8496	2768.54
Proposed MCDM	**-0.035**	**699.60**

Table 4.6: Model Prediction Accuracy for Categorical Fields

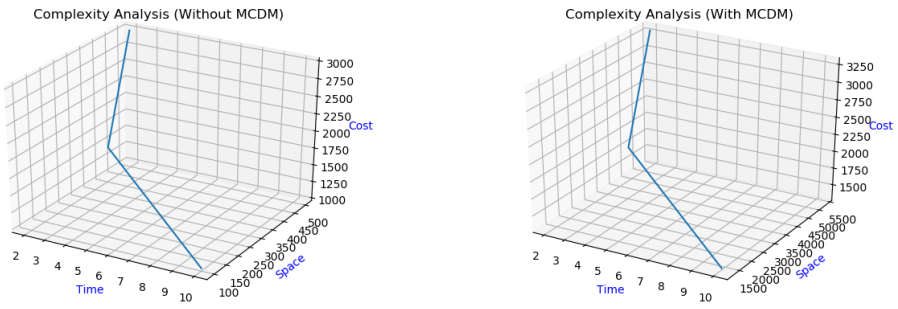
Algorithm	**Precision**	**Recall**	**Accuracy**	**F1 Score**
Gaussian NB	0.10	0.17	0.17	0.13
Bernoulie NB	0.36	0.52	0.52	0.43
Decision Tree Classifier	0.45	0.50	0.50	0.47
SVM	0.76	0.71	0.71	0.73
Logistic Regression	0.34	0.48	0.48	0.40
Proposed MCDM	**0.91**	**0.89**	**0.89**	**0.90**

4.8 Discussion

MCDM produces better insights than one-framework fits-all models by employing different methods for each discrete set of data. Nevertheless, in the online mode of execution, MCDM induces computational overhead and causes latency in the response time. MCDM is placed in between

Predicted actual	Predicted 7	Predicted 8	Predicted 9
7	2 (100%)	0 (0%)	0 (0%)
8	0 (0%)	1 (50%)	1 (50%)
9	0 (0%)	0 (0%)	5 (100%)

Figure 4.10: Confusion Matrix for MCDM categorical forecasting. Along the diagonal cells, counts for the correct predictions are placed, counts for incorrect predictions are placed in the rest of the cells.



(a) Without MCDM

(b) With MCDM

Figure 4.11: Time, space and cost complexity with MCDM decision making component. MCDM induces 5% delay in the response time. The diagrams shows complexity with MCDM (Figure 4.11b) and without MCDM (Figure 4.11a). Space usage and cost is higher in MCDM online mode since all the alternative tools are developed/purchased and running simultaneously in the cluster. The MCDM requires at least 30% more storage space.

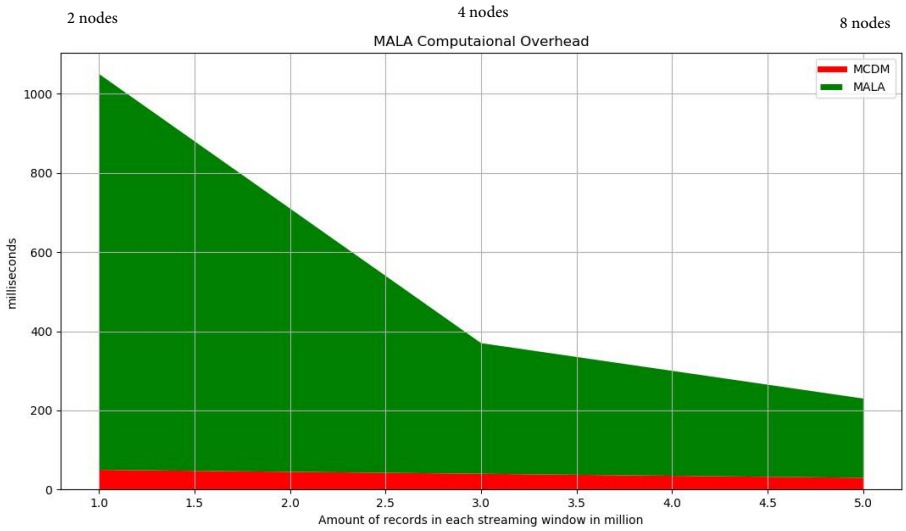


Figure 4.12: Computational overhead due to MCDM. In this Stackplot, the green area shows the processing time by MALA, and the red area is the latency due to MCDM. MCDM induces a 5% delay in response time on average. MCDM latency decreases slightly when the number of nodes in the cluster increases, and MCDM is installed into multiple nodes along with a load balancer.

incoming data streams and the processing engine. In the online setting, MCDM creates the graphic structure on every window length of ingested data based on the data schema or property and requirement criteria. It then runs the RDF Query Language to select the best-fit methods, leaving an amount of delay. However, the selection process is based on the metadata and not on the large volume of actual data. RDF Query Language is designed for faster graph processing to ensure minimum impact on overall responsiveness.

An evaluation of latency overhead due to MCDM (Figure 4.12) was conducted. MCDM induces a 5% delay in response time on average. Apart from causing a delay in response time, the MCDM needs more storage space to operate and incur a higher cost as all the alternative components need to run simultaneously for MCDM to choose the best options. Figure 4.11 compares time, space, and cost complexity (3-D) without and with MCDM. Besides 5% latency, MCDM can cause 30% more space usage and subsequent infrastructure cost, at minimum.

Processing delay led by an additional layer of the MCDM decision-making component causes

events to be processed based on the time they reach the processing engine. As processing time may differ significantly from event generation time, it can result in an undesirable outcome. However, the watermarking techniques [245] [52] can be employed to process the late-arriving events based on event generation time.

Despite the negative impact of MCDM due to time, space, and cost overheads, the MCDM can be still worthy of applying in multi-source, heterogeneous big data computing environments. Higher accuracy and precision with MCDM's benchmarking results from Table 4.5, 4.6 may offset the negative factors. Also, in the offline mode, the MCDM does not cause any additional space or time complexity, which is only applicable to online decision-making scenarios.

4.9 Summary

This chapter presented a multi-criteria decision-making framework through a graph-based approach. The proposed MCDM is the *second layer* in the multimodal data processing pipeline the thesis introduces. The decision-making component is placed as a gateway of the pipeline between ingestion and downstream processing components. A graph-based framework was demonstrated for an offline and online decision-making system in a strategic interaction of a society of multiple criteria. In the online mode, the framework can facilitate a single model to process a heterogeneous data pool (variety) as well as a high rate of data ingestion (velocity). Hence, the proposed framework is worthy of implementing a big data analytics pipeline. In the offline mode, the framework helps the benchmark between a set of alternative problem solvers without actually installing them. Therefore, the offline mode is cost-effective and precise in decision-making, improving the existing standalone algorithms and methods. The benchmarking results demonstrate that the proposed system significantly improves the performance of existing algorithms in the big data scenario through its heterogeneous, multi-criteria data processing capabilities.

Chapter 5

Incremental Lifelong Learning

5.1 Introduction

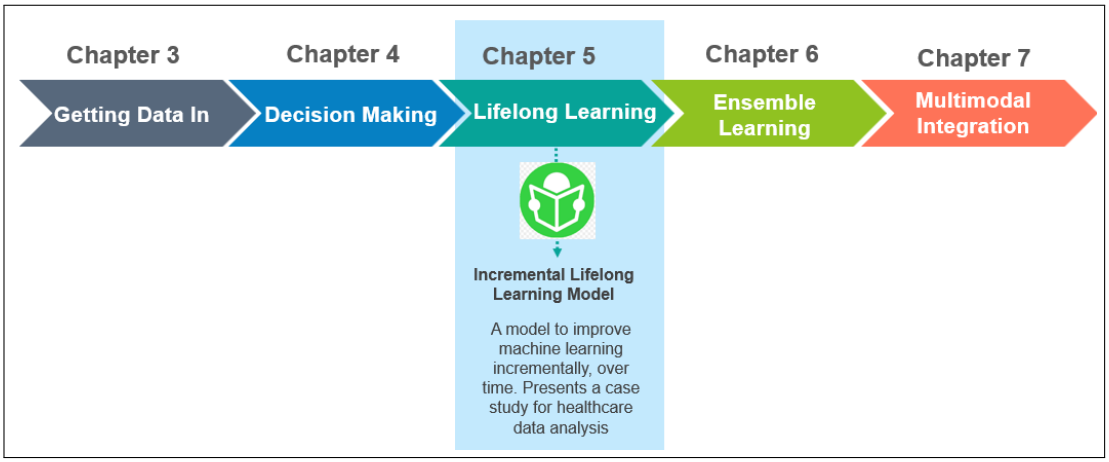


Figure 5.1: An outline of Chapter 5 and end-to-end data flow pipeline in different chapters.

The previous two chapters dealt with data *ingestion, in-flight processing* and real-time *decision-making* paradigms selecting the best-fit methods for each incoming data stream. As the next component in the analytical pipeline, this chapter presents a *Lifelong Learning* model at the processing layer after data is acquired, and a set of tools are selected for the *downstream processing* (See

Figure 5.1). Using past acquired knowledge to improve future learning leads to the next generation of machine learning. Lifelong learning mimics the *always learning* experience of human life [277] [285]. Training a large volume of historical datasets requires a large cluster and expensive infrastructure setup, making the long-running one-shot batch jobs unrealistic on a big data pile. Batch tasks are not agile enough to adapt fast with each new wave of ever arriving datasets. Hadoop and the de facto *MapReduce* framework make the learning process wait until the full set of data are collected and hours or even days long batch jobs get completed. On the contrary, streaming learning enables models to be trained and updated after each passing mini-batch of the window. Therefore, the model can catch up to the newest trends faster and with a much smaller cluster size, reducing the infrastructure overhead significantly. To address the *responsiveness* issue of the traditional batch model, this chapter introduces a hybrid approach through big data lambda architecture, which considers batch and stream as two collaborating agents in a multi-agent system.

An era of weak AI. Over the past 30 years, machine learning has achieved significant development. However, in many ways, we are largely still in an era of *Weak AI (or narrow AI)* rather than *Strong AI* (or broad AI) [75]. Current machine learning algorithms have the efficiency in solving a specific problem, but they are not efficient enough to *reuse* the past acquired learning on a group of *related problems*. Hence, lifelong machine learning (or, simply put, Lifelong Learning) [249] was developed to solve analyzing an infinite sequence of related tasks by knowledge accumulation and reusing. For a group of related problems, an integrated model with knowledge reusing could decrease the overhead for data sample annotation. Consider the scenario of *sentiment classification* for predicting the sentiment polarity (positive or negative) in a sentence or a document. For different sentiment classification tasks, traditional approaches need to train an independent model on each domain to obtain the best performance. Hence, for each individual domain, we need to collect labelled data for supervised learning. In this way, the algorithm can not solve a problem by reusing the past learning and needs further *labelled data*. This can be typically described as *Weak AI*.

Lifelong learning approach. The proposed Lifelong Learning framework continuously adapts

to changing data patterns over time using an incremental learning approach and by knowledge reuse. In many big data systems, iterative retraining of high dimensional data from scratch is computationally infeasible since constant data stream ingestion in addition to the existing historical data pool increases the training time exponentially. Therefore, the need arises to retain past learning and quickly update the model incrementally based on the new data. Also, the current machine learning strategies provide model prediction without a comprehensive Root Cause Analysis (RCA). The proposed framework lays foundations on the *ensemble process* between stream data with historical batch data for an Incremental Lifelong Machine Learning (ILML) model to resolve these limitations.

Motivation. Training a large volume of historical data requires large clusters and an expensive infrastructure set up, making the long-running one-shot batch jobs unrealistic on a big data pile. Batch tasks are not agile enough to adapt quickly with each new wave of ever arriving datasets. Hadoop and its de facto MapReduce framework make the learning process wait until a complete set of data is collected and hours or even days long batch jobs get completed. On the contrary, streaming learning enables models to get trained and updated after each passing mini-batch of a window. So, the model can catch up to the newest trends faster and with a much smaller cluster size, reducing the infrastructure overhead significantly. However, streaming learning can get overwhelmed with a large historical data pool, to begin with the process. To address the *responsiveness* issue to the traditional batch model, this chapter introduces a mix-model approach through a big data lambda architecture that considers batch and stream as two collaborating multi-agent systems. To start with, the batch system saves its learning to the Hadoop Distributed File System (HDFS) for later use by the stream engine. Streaming systems can update the model incrementally. The design protects streaming learning from getting overwhelmed by a large set of static historical data pools. The streaming model initializes itself with *saved learning* from the batch by loading the trained model from HDFS into the Apache Spark Discretized Stream (DStream). A configurable mini-batch time window (of say 1 hour) retrains the model. At the same time, the model can predict test data continually. The updated model by stream engine is persisted and replicated into HDFS

at a periodic interval (say, six hours) to run any ad-hoc batch query. Post-processing, additional static, historical data pool can be merged with results from the stream engine, and stream processor can *pick and continue* from thereon processing in a sliding window.

5.2 Organization of this Chapter

The organization of this chapter is as follows. The high-level objective of the current work is to develop a general-purpose architecture for Lifelong Learning using an incremental learning approach. Section 5.3 highlights the current research gaps this work intends to address. Section 5.4 outlines the key contributing areas in the field of Lifelong Learning, sentiment classification, streaming clustering, dimension reduction, and Random Decision Forest-based forecasting and root cause analysis. An overview of two case studies about the proposed lifelong machine learning model is presented in section 5.5. The proposed system is based on successive streaming clustering and Random Decision Forest Regressor and Classifier, which uses Multi-agent Lambda Architecture (MALA), an ensemble framework between the stream and batch data. MALA is described in section 5.6, followed by the streaming incremental learning method presented in Section 5.7. A case study with prostate cancer patients' data is presented in sections 5.8. Section 5.9 offers an approach to sentiment polarity using Lifelong Learning and crawled data from Amazon.com.

5.3 Current Research Gaps

Through the period of the last two decades, there has been significant progress in machine learning algorithms and frameworks. However, much less emphasis was given on how these methods and algorithms can be used to train over an extended period to incrementally become more efficient through knowledge retainment and transfer. This chapter addresses these research gaps by developing an incremental, transfer learning model through a big data stream-batch mix processing approach. A novel collaborative mixed processing framework called Multi-agent Lambda Architecture (MALA) model is presented, which uses Hadoop MapReduce at the batch layer and

Apache Storm at the stream layer to develop a *Lifelong Learning machine*. Also, the current predictive analysis frameworks provide forecasting without much explanation and root cause analysis. The work presented in this chapter fills the research gaps between only analytics and more explicit *actionable insights*, through reasoning and a Lifelong Learning platform. The framework is effective in high dimensional big data applications using Apache spark in-memory data processing capability, dimension reduction techniques, and never-ending incremental learning.

5.4 Summary of Contribution

The chapter addresses the Research Question 3 about *the multimodal collaborative model between batch and stream processing that produces a more accurate and comprehensive representation of data compared to a single standalone knowledge base.*

1. **Lifelong Machine Learning model** This work presents an incremental LML model through Multi-agent Lambda Architecture (MALA), a collaborative ensemble framework for the stream and batch data. Initializing with batch processing results as an offset, the framework successively applies: (i) a streaming clustering to group the data points and isolates the target data points, followed by (ii) a Random Decision Forest Regressor and Classification algorithm provide reasoning through dynamic root cause analysis on the focus groups, compares between groups and performs forecasting to demonstrate the efficiency of the proposed LML model. The model's real-time and batch agents use Spark MLlib APIs. In the MALA framework, a Knowledge Miner (KM) consolidates the past learning with recent stream updates into a Knowledge Base (KB), transforming the model into a Lifelong Learning System.

2. **Sentiment Classification** An alternative model is developed for *sentiment classification* using the LML approach. Computing *sentiment polarity* is a two-step process. The *first step* is the initial learning stage building on top of a large volume of a historical dataset at a *batch setting*. The second step is a *self-study* stage where a previously trained model can apply itself for self-learning and prediction in a *streaming* setting.

Natural Language Processing (NLP) is a preferred field for lifelong machine learning researches. Previous works [64] chose the sentiment classification as the learning target, where a large task was divided into a group of related sub-tasks in the different domains. These sub-tasks are related to each other, but a model *trained only on a single sub-task* underperform in the remaining sub-tasks. Therefore, algorithms require to know when the prior knowledge *can be used* and *when can not*, due to the different distribution of each sub-tasks. Thus, the proposed algorithm can be called *lifelong* because it *transfers* previous knowledge to new tasks to improve performance in a never-ending cycle of the learning process.

3. **Streaming clustering through dimension reduction** The objective is to achieve clusters of similar data profiles and isolate the specific groups for further investigations. The fundamental idea for an online version of K-Means clustering is to divide the data stream into mini-batch windows and to incorporate knowledge learned in the previous window into the following ones. Hence, in streaming K-means clustering, the model is updated with each rolling window based on a combination of cluster centres computed from the preceding mini-batches and the current mini-batch. The framework further enables merging a large static historical data pool with the latest and most updated streaming model. A unique dimension reduction method is adapted for the feature vector to reduce the model training time significantly. The dimension reduction method of feature vectors enables quick retraining for clustering and Random Decision Forest through the MALA framework. With this approach, the batch model creates the offset (or initial point). On each streaming window of the dataset, the streaming model only needs to learn the updated features or *corrections* during the retraining process. The contribution addresses the critical research question about the collaborative model between batch and stream processing.

4. **Random Decision Forest-based forecasting and root cause analysis** A Random Decision Forest Regressor graph drills down into each anomalous cluster and provides a comprehensive root cause analysis and prediction for future items. The proposed framework produces the Decision Tree graph in JSON formatted data as well as in a tree structure. Several op-

timizations are proposed for the Random Decision Forest model, including data sampling, important feature selection, and optimal Decision Tree hyperparameters to improve the training time using reduced data volume without affecting the overall accuracy.

The two-stage process of the proposed architecture is depicted in Figure 5.2.

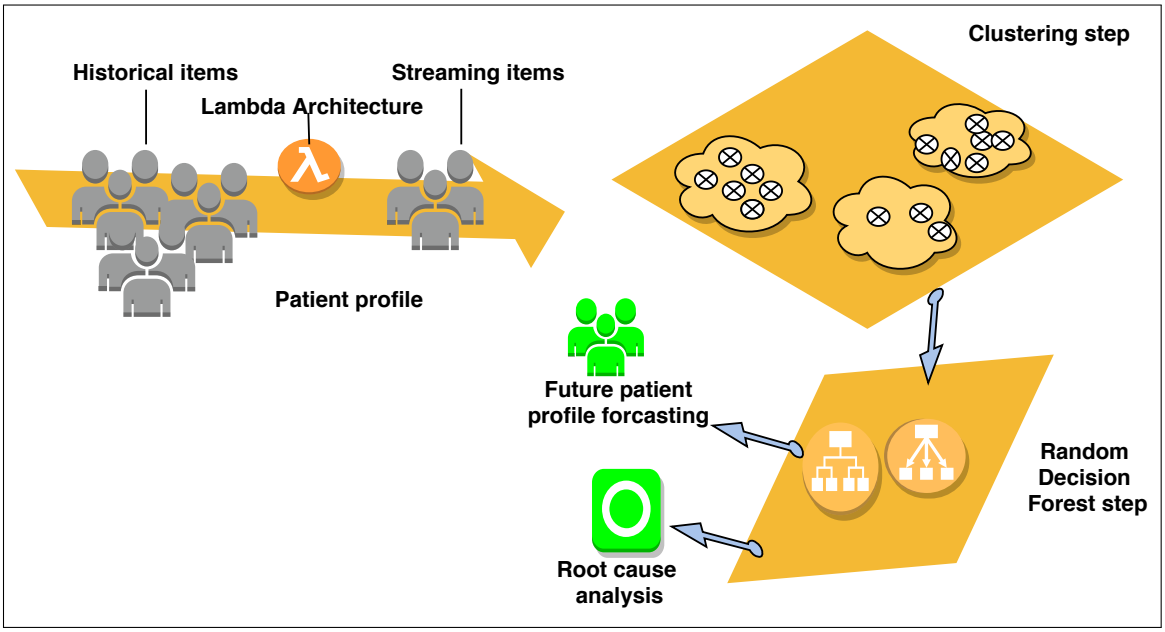


Figure 5.2: **The End-to-end flow of the proposed architecture.** The method proceeds in two stages. (i) A hybrid clustering creates a similar data profile and detects outliers. (ii) A Random Decision Forest process creates a decision graph on each anomalous cluster providing reasoning with root cause analysis and forecasting on future items.

5.5 Case Study Description

An overview of the case studies is presented here, describing the capabilities of the proposed Lifelong Learning model. The case study with prostate cancer patients contains a large volume of high dimensional data where each record has about 20000 features.

1. **Prostate Cancer Patient Data.** A case study is presented on real-world cancer patient data. A cancer patient's pathological tests like blood, DNA, urine, or tissue analysis reveals a

unique signature based on the DNA combinations. Results and analysis of clinical data help to prescribe personalized and targeted medications and achieve a therapeutic response. The model presented in Figure 5.2 is evaluated through data from The National Cancer Institute's Genomic Data Commons unified data repository. The objective is to prescribe personalized medicine based on the thousands of genotype and phenotype parameters for each patient.

2. **Sentiment Classification.** A novel approach is presented through Lifelong Learning to discover texts with sentiment classification of unstructured textual data (Amazon.com product reviews) and reuse the learning in future predictions. The case study reveals the probability of each word showing in the positive or negative content within the corpus of textual data consisting of product reviews. The case study also identifies the words only having sentiment polarity in some specific domains (equal to tasks in this chapter). *Lifelong Sentiment Classification (SLC)* [64] identifies the domains which have words with sentiment orientation. If a word always has sentiment polarity in the current domain, a higher weight is assigned to it than in other words in the corpus. The case study demonstrates the efficiency of LML with respect to Natural Language Processing with high *data volume*.

5.6 Proposed MALA Ensemble Framework for Lifelong Learning System

The proposed method is based on successive streaming clustering and Random Decision Forest Regressor and Classifier that uses MALA, a multi-agent ensemble framework. MALA is described in this section; in the following sections, clustering and Random Decision Forest algorithms are discussed.

MALA is a consolidation framework between stream and batch modules based on the fundamentals of LA. MALA is developed as an extension to the standard LA framework, and the MALA framework is the main contribution that extends this line of work. The framework initializes itself with batch processing results for the streaming engine to incrementally build on top of the batch

offset. The framework successively applies a streaming clustering in the current context, followed by a Random Decision Tree algorithm to provide dynamic root cause analysis and forecasting. The framework enables collaborative, accumulative learning through big data tools and APIs described in the thesis. Streaming and batch components are two cooperative, autonomous agents in a multi-agent system. The following are the *three* components of the MALA:

Historical Data Store (HDS). In a batch setting, a large volume static data pool is ingested, all at once and at a periodic interval, processed and written back into a disk, using frameworks like Apache Hadoop and Spark. The HDS stores preprocessed unstructured data accumulated over time in a distributed file system such as HDFS, NoSQL databases, or Amazon Simple Storage Service(S3). The model is trained on the entire data pool. In batch mode, the response time is never a primary constraint, but rather the design is inclined towards a comprehensive coverage of the data. Historical Data Store (HDS) persists the trained models in the file system on in a database.

Stream Processor (SP). The SP updates the training model on each new wave of incoming data. The streaming process initializes itself with *saved learning* from the batch by loading the previously trained model from persistent distributed storage into distributed memory. The continuous data stream updates the model *incrementally* compared to *one-shot* batch job. The training time largely depends on the mini-batch data size and window length. The duration can vary from a few milliseconds to hours. Apache Spark Streaming creates in-memory *DStreams* from the stored model in HDFS produced by the batch jobs. After each iteration of retraining, the updated model is persisted into memory and disk. Since the training data volume keeps getting larger, the most recent data is cached into distributed memory, and the remaining is stored on a disk. The amount of distributed memory size is configurable, for example, using the Spark configuration file. Stream processing includes filtration of data rows and converting, transforming the ingested flow into structured data.

Knowledge Miner and Knowledge Base. Knowledge Miner (KM) consolidates the past learning with recent stream updates into the Knowledge Base (KB) as a database of records in a structured format. KB does not deal with original data as in HDS and SP but only stores the

post-processing results such as trained models. KM is responsible for filtration (eliminating noisy and faulty records), knowledge aggregation, and data governance for monitoring and reporting purposes. Filtration and aggregation logic is ad-hoc and developed for a specific case study. At the implementation level, KM schedules the batch jobs (i.e. Spark, MapReduce), which create the initial training model and spawns the Spark streaming jobs to iteratively retrain the model. KM builds the Decision Tree from JSON formatted data and provides visualizations and predictive analytics results by ad-hoc queries through a web-based search interface.

KM consists of a *Amazon Route 53* Domain name System (DNS), providing a fault-tolerant routing system [38]. *Rout 53* effectively translates domain names into IP addresses. Another important feature of Route 53 is the *geolocation routing*, so that requests from Europe are sent to a European data centre and so on, reducing the latency due to network travel.

KB features a shared storage option between stream and batch modules. As shown in Figure 5.3, results from the batch are merged into the steaming output to provide a comprehensive view of data and removing any cold start situation. Subsequently, stream processing results *influence* batch output as illustrated with a case study for the Recommender System in Chapter 6.

5.7 Streaming Incremental Learning

Incremental learning in the MALA ensemble framework is explained using a *two* steps process: streaming clustering, followed by Random Decision Forest Regressor and Classifier. The framework initializes with the batch operation results as an *offset* or the starting point. Thereafter, each mini-batch window of stream data retrains the machine learning model iteratively. Since data volume grows over time, retraining the entire system on each streaming window becomes computationally infeasible. Therefore, the proposed framework selectively updates the training model through a dimension reduction method, avoiding retraining the entire system.

Problem Statement (Cumulative Hybrid Learning): At time t, the system maintains a set of cluster $C^t = C_1, C_2,..,C_n$ with training model M^t from a past dataset. Once enough training data is collected on a mini-batch window interval δ_t, M^t is updated to cover new training model M^{t+1}.

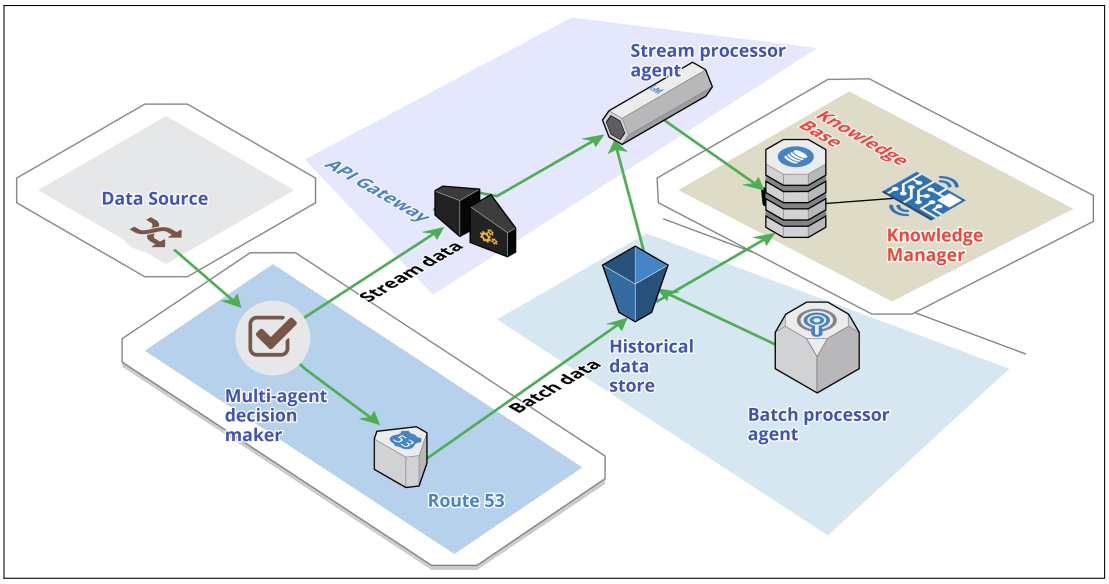


Figure 5.3: Multi-agent Lambda Architecture (MALA). Four layers of MALA comprise the historical data store, stream processor, Knowledge Base, and Knowledge Miner. Real-time and batch layers are autonomous Multi-agent systems in collaboration. MCDM, a Multi-agent decision maker component, is placed into the MALA stack at the gateway of the data pipeline for further processing. Once the decision-maker component selects the best-fit method for the most recent dataset, data moves to the real-time processor as well as the HDFS storage for batch processing. The Knowledge Base consolidates both the views, which acts as the serving layer for the user query.

The objective is building M^{t+1} with minimum latency by selectively retraining part of the dataset.

Lifelong Machine Learning (LML) through MALA: At any point of time t, the learning system has performed n training tasks $\tau_1, \tau_2, ..., \tau_n$ on past datasets $\mathfrak{D}_1$,$\mathfrak{D}_2$,...,$\mathfrak{D}_n$. On arrival of the new streaming set $\mathfrak{D}_{n+1}$ the training system uses the past acquired knowledge stored in the Knowledge Base to help learn the current task τ_{n+1}.

After accumulation of knowledge τ_{n+1}, the *Knowledge Base* is iteratively updated (Figure 5.3). The update involves selective retraining, removing inconsistency, and updating metadata. The learning system updates the current model M^t to M^{t+1}, that can cluster new data $\mathfrak{D}_{n+1}$ into an existing old cluster or an unseen new cluster. The steps involved are summarized as follows and sequentially described:

1. **Map dataset to a cluster.** Searching for the cluster in C^t similar to new dataset $\mathfrak{D}_{n+1}$. Create a separate cluster if data points are not fitting into the exciting clusters in C^t.

2. **Streaming clustering step**. An incremental clustering procedure groups the dataset by the dimension reduction method.

3. **Decision Tree and Random Decision Forest Step**. The Random Decision Forest algorithm further drills down into each group to compare groups and provides reasoning about anomalous clusters through each Decision Tree graph in Decision Forest and predicts future elements.

5.7.1 Streaming clustering step

Under the streaming setting, in the bounded small space S, data is accessed through a linear scan, i.e. only once and viewed in order. The batch version of the K-means algorithm provides an offset (or, initial point) for the streaming learning to update the model iteratively.

Algorithm:

1. Divide S into n disparate pieces a_1,...,a_n. number of cluster

2. Find $\mathcal{O}(k)$ centers in a_i, allocate each element in a_i to their closest cluster center according to the Euclidean distance function.

3. Let C is the data points for which $\mathcal{O}(k)$ center computed in step (2), while each center $c \; \varepsilon \; C$ is weighted by a number of points allocated to it.

4. Data points C gets k number of clusters.

5. For every streaming window, re-compute new cluster centers using:

$$c_{t+1} = \frac{c_t \; n_t \alpha + x_t m_t}{n_t \alpha + m_t} \tag{5.1}$$

$$n_{t+1} = n_t + m_t \tag{5.2}$$

   Where n_t is the historical data set and c_t is the old cluster centroid. m_t is the newest data points and x_t is the cluster center computed with new data points. α is the decay factor. With α=0, only the latest window dataset is used; for α=1, an entire historical dataset is used.

The online version of K-Means clustering is to divide data stream into mini-batch windows and to incorporate knowledge learned in the previous window into the following ones. Hence, in streaming K-means clustering, the model is updated with each rolling window based on a combination of cluster centres computed from the preceding and the current mini-batches. The streaming algorithm is adapted from the most recent release of Apache Spark [11]. The Apache Spark MLLib [12] libraries include a streaming version K-means clustering. The algorithm starts with initializing data points to their closest clusters. For every iteration, with new data arrival, the method evaluates new cluster centres, then updates each cluster using Equation 5.1.

5.7.2 Decision Tree and Random Decision Forest Step

A collection of Decision Trees is generalized to form a more powerful algorithm as Random Decision Forests. The flexibility of Random Decision Forests makes these worthwhile to examine the dataset. For the number of n features in the dataset, there is a $2^n - 2$ possible decision rule (all

subsets except the empty set and entire set). This leads to the possibility of creating a few billion decision rules for even moderate size data. The decision rules use several heuristics to select the top n decision rule as described below in the steps for constructing a Random Decision Forest. The key contribution in the method described in this step is the novel Lifelong Learning and dimension reduction techniques using the batch-stream interaction as described below:

Step 1. Sampling training subsets

In batch mode, n subsets are sampled from the raw training dataset S using Spark MLlib stratified sampling methods [13], providing a fraction of sample size and a random seed. As an instance, from the patient dataset, men and women patients can be sampled as gender being the *key* in *sampleByKey()* method.

Step 2. Important feature selection

In batch mode, we can decide how much each feature contributes to the final prediction through Spark *featureImportances* API [13]. A list of the important features is selected over the sampled dataset in the previous step. Within the scope of current case study, for a tree ensemble model, the function Spark API *RandomForest.featureImportances* [14] finds the importance of each feature from about 20000 *genotype* and *phenotype* features associated with each record in the training dataset and eliminate a significant number of unrelated parameters. The method generalizes the concept of *Gini Importance* for approximating the independent target variable.

Step 3. Dimension reduction of feature vector

A unique dimension reduction method retrains the feature vector faster. The batch model creates the offset (or initial point). The streaming model needs only to learn the updated features or corrections during the retraining process on each streaming window of the dataset. Let p_{it} be the feature vector of item i at time t and let q_{it} be latent factor vector of item i at time t. Then expected training

model M at time t for item i is:

$$M_{it} = p'_{it}A + p'_{it}B + p'_{it}q_{it} \tag{5.3}$$

Where A and B are the regression weight matrix learned through the offline training process. $p'_{it}A + p'_{it}B$ provides an initial offset for the online training process. $p'_{it}q_{it}$ is learned through the online process. Suppose, q_{it} consists of a k-dimensional linear subspace extended by the columns of a random projection matrix $B_{r\times k}$ $(r \gg k)$ that is,

$$q_{it} = B\delta_i \tag{5.4}$$

The online component only learns a k-dimensional vector δ_i. To eliminate numerical ill-fitting during computations let us assume, $\delta_i \sim MVN(O, \sigma_\delta^2 I)$
where MVN denotes the *Multivariate Normal Distribution* [34] and σ^2 is the variance of noise in actual data. While marginalizing over δ_i, it is evident that $q_{it} \sim MVN(O, \sigma^2 BB')$. Because $rank(B) = k < r$, then the probability distribution in a lower-dimensional subspace can be extended by the columns of B. It is usually better to go with a model which assumes $q_{it} = B\delta_i + \varepsilon_{jt}$, where $\varepsilon_{jt} \sim MVN(O, \tau^2 I)$ is an error of the model after the k-dimensional linear projection. The process eliminates the rank inadequacy and the distribution of $q_{it} \sim MVN(O, \sigma^2 BB') + \tau^2 I$. Despite the model's easy compatibility with Gaussian response, due to the model's computational overhead to the current model, assume $\tau^2 = 0$. So, $q_{it} \sim MVN(O, \sigma^2 BB')$.

Step 4. Constructing the Decision Tree

The Decision Tree is built using the *C4.5* algorithm. Primary improvements with *C4.5* over the *ID3* algorithm are the pruning technique and the capability of handling continuous attributes, null values, and noisy data [111] [214]. In the building process of the tree, for each streaming window, n important features (in total) are selected in the previous steps 1 and 2. Let us assume a feature f_i is selected with the highest normalized information gain on splitting on the feature as the root.

Once a feature f_i is selected as the root of a node, on each streaming window, the remaining nodes create the children by recursively calling C4.5.

Algorithm 4 Lifelong Learning through Dimension Reduction and MALA Ensemble Framework

Input: *S:* Training dataset
n: Input feature variable
k_t*:* Kafka topic
d_b*:* historical batch datastore
$d_s = \sum_{i=1}^{n} d_i$: dataset as a collection of streaming records since the last window
Output: M: Final trained model
KB: updated Knowledge Base
1: Reduced sample dataset $S' \leftarrow$ sample(S, *fraction*, *seed*) for the sample dataset S
2: **for** each input feature f_i in S' **do**
3: $f_i \leftarrow$ importantFeatures(f_i) //top n important feature as a list
4: **end for**
5: KB $\leftarrow$ batchComputationRule(d_b)
6: subscribeKafka()
7: **for** windowed dataset d_s **do**
8: Spark consumes Kafka queue
9: Update model $M_{it} = p_{it}A + p_{it}B + p_{it}q_{it}$, $q_{it} = B\delta_i$. /*retrain the model, update only δ_i through the online process, rest through the batch*/
10: M $\leftarrow$ M + M_{it}
11: $KB \leftarrow$ streamComputationRule(δ_i) /*constructing the Decision Tree, streaming clustering etc*/
12: $KB \leftarrow M$
13: **end for**
14: **return** KB

Sampling *n* training subset (line 1): n subsets are sampled from the raw training dataset S using the Spark RDD sampling method. For instance, the dataset can be sampled based on gender, age group, or patient genetic type.

Extracting important features (lines 2-4): Find a list of the important features over the sampled dataset. For example, the current training dataset contains around 20000 genotype and phenotype features associated with each patient. This step reduces the number of features to a few hundred based upon the contribution of features to the interdependent variable and removing insignificant

features.

Apply batch computation rule (line 5): The batch computation rule is applied to the historical dataset to update the Knowledge Base and initializes the stream processing engine. Knowledge Base is updated subsequently with the batch computation results.

Kafka message queue (line 6): Kafka is the message queue subscribing to the stream processing engine.

Return the updated Knowledge Base (lines 7-12): *Spark* consumes the *Kafka* queue as a series of *DStreams* and retrains the model with the dimension-reduced feature vector. The Knowledge Base is updated subsequently with the latest model and stream computation results.

Return the updated KB (line 14): Return the updated Knowledge Base to the serving layer for the user queries.

The Random decision forest prediction is based on the weighted average of each tree's predicted values. In the dataset with prostate cancer patients, the Random Decision Forest prediction is based on the weighted average of the probable values from each tree and computing the most likely value out of it. The Spark *RandomForestClassifier* API builds the Random Decision Forest model for categorical features on the implementation level. Spark *RandomForestRegressor* API are used for numerical features. See Table 5.4, 5.5 for detailed results for numerical and categorical features. The algorithm requires all feature vectors collected into one column. *VectorAssembler* is used as a DataFrame transformer within the *Spark MLlib pipelines* API to build the initial Decision Tree feature vector as the training dataset. The classifier model's return type is *DecisionTreeClassificationModel*, after MLlib APIs transforms the data into frames and the return value contains predictions objects. A *MulticlassClassificationEvaluator* computes the accuracy of classifiers with the *F1* score representing a weighted average of precision and recall. A *confusion matrix* is also produced for the quality of the classifier's output. A *confusion matrix* is $n \times n$ where n is the number of possible outcomes. Values across the rows representing real values, while predicted values are placed along with columns. The entry at row i and column j represents a count of the number of times an instance with correct category i was predicted as category j. Therefore, the number

of correct predictions placed along with the diagonal cells and counts for incorrect predictions are placed in the rest of the cells. See Figure 5.14.

5.7.3 Tuning Decision Tree Hyperparameters

The following model hyperparameters are tuned to fetch the optimal result: *maximum depth, maximum bins, impurity measure, and minimum information gain*. Maximum depth is the most number of connected decisions the classifier takes. The value for maximum depth is selected after several trial runs on the training set to prevent model overfitting. Bin size is proportional to cluster size, with a large bin size leading to an optimal decision rule. The model uses Gini impurity [242] which is directly related to the accuracy of the random-guess classifier computed through the following equation:

$$I_G(p) = 1 - \sum_{n=1}^{n} p_i^2 \tag{5.5}$$

where p_i is the proportion of elements of class i out of n available classes. Minimum Information Gain rejects candidate decision rules that do not improve the subsets' impurity, thus resisting model overfitting. A number of combinations of the hyperparameters are tried, and corresponding results are reported. The model is trained and evaluated with *two* probable values of each hyperparameters producing *16* models in combination. Each model is evaluated by *MulticlassClassificationEvaluator* in Spark MLlib APIs [15] to compute the evaluation results for each of the combination of model hyperparameters in terms of *accuracy* and *F1 Score*. The objective is to find the optimal values for *maximum depth* and *maximum bins*, which should be sufficient enough but not excessive to improve the training time and could resist overfitting. *getEstimatorParamMaps* [110] API provides accuracy of each combinations of hyperparameters in a single run. Therefore, avoiding potential thousands of independent test runs for each combination. Note, *CrossValidator* APIs can also be used for n-fold cross-validations. Nevertheless, it also induces n times more computation complexity and unsuitable for big data applications. The tests select the best combinations for model hyperparameters. The tests reveal that the Gini impurity works best at a max depth of *30*,

along with *55 bins*. See Figure 5.4 showing the sample screenshot of the tests.

```
0.9278058059438129
{
      dtc_3cdb9fdl001b-impurity: gini,
      dtc_-3cdb9fdl001b maxBins: 30,
      dtc_3cdb9fdl001b -maxDepth: 20,
      dtc_3cdb9fdl001b -minInfoGain: 0.0
}

0.9412387096518764
{
      dtc_4bc56098fa05-impurity: gini,
      dtc_4bc56098fa05-maxBins: 55,
      dtc_4bc56098fa05-maxDepth: 30,
      dtc_4bc56098fa05-minInfoGain: 0.0
}
```

Figure 5.4: **Sample screenshot of the tests.** The test finds the combination of model hyperparameters to produce the best accuracy.

5.8 A Case Study with Prostate Cancer Patient Data

5.8.1 Application scenario

The National Cancer Institute's Genomic Data Commons (https://gdc.cancer.gov) released a unified data repository for the cancer research community in support of personalized medicine. The repository contains over 32000 patient cases, including clinical data, treatment data, biopsy results, gene expression data, as well as a wide range of other information [16]. Thus, allowing accessibility to researchers to uncover novel biomarkers and find a correlation between genes and survival.

Personalized cancer medicine is customized to each patient's need for chemotherapy or drugs based on patients' specific set of DNA or genes. A cancer patient's pathological tests like blood, DNA, urine, or tissue analysis provide a unique signature based on the DNA combinations.

Results and analysis presented here aim to contribute to improving the overall survival rate for patients with prostate cancer through personalized and targeted medications and achieves a therapeutic response based on the thousands of genotype and phenotype parameters.

5.8.2 Metadata and model training

The patient repository contains over 32000 anonymized patient cases, including clinical data, treatment data, biopsy results, gene expression data, as well as a host of other pathological information, allowing us to uncover novel biomarkers and find a correlation between genes and survival. Each patient data comprises thousands of biometric data for each patient's DNA/RNA sequencing, gene expression, molecular profile, mutational status, drug therapies, survival length, physical characteristics, etc. There are about 21000 parameters associated with each of the patients, the majority of which is sourced from gene expression data (>20,500), and the remaining are other pathological identifiers (>100). For example, primary human genes influencing prostate cancer patients' overall survival rates are gene HOXB13, MSMB, and CDH1. The gene expression level in each patient varies, with MSMB differing between 6 to 15 among all patients in the database. Refer Figure 5.5.

The proposed model selects the priority of each feature from about 20000 genotype and phenotype parameters linked with each record in the training dataset and eliminates a significant number of unrelated parameters. Important features as extracted by the model are listed in Tables 5.1 and 5.2.

Table 5.1: Important genes extracted by the model

Column labels: Common genes associated with cancer							
AR	CHEK2	EZH2	HOXB13	LRP2	MXI1	PTEN	BRCA1
EHBP1	FGFR2	MAD1L1	NBN	RNASEL	WRN	BRCA2	ELAC2
FGFR4	IGF2	MED12	SRD5A2	WT1	CD82	EP300	GNMT
ITGA6	MSMB	PCNT	STAT3	ZFHX3	CDH1	EPHB2	HNF1B
KLF6	MSR1	PLXNB1	TGFBR1				

Table 5.2: Important parameters extracted by the model

Column labels: Other pathological identifiers	
Overall survival (OS)	Pathologic T
Biochemical recurrence	PSA value
Days to first biochemical recurrence	Additional radiation therapy
Gleason score	Sample type
Pathologic N	

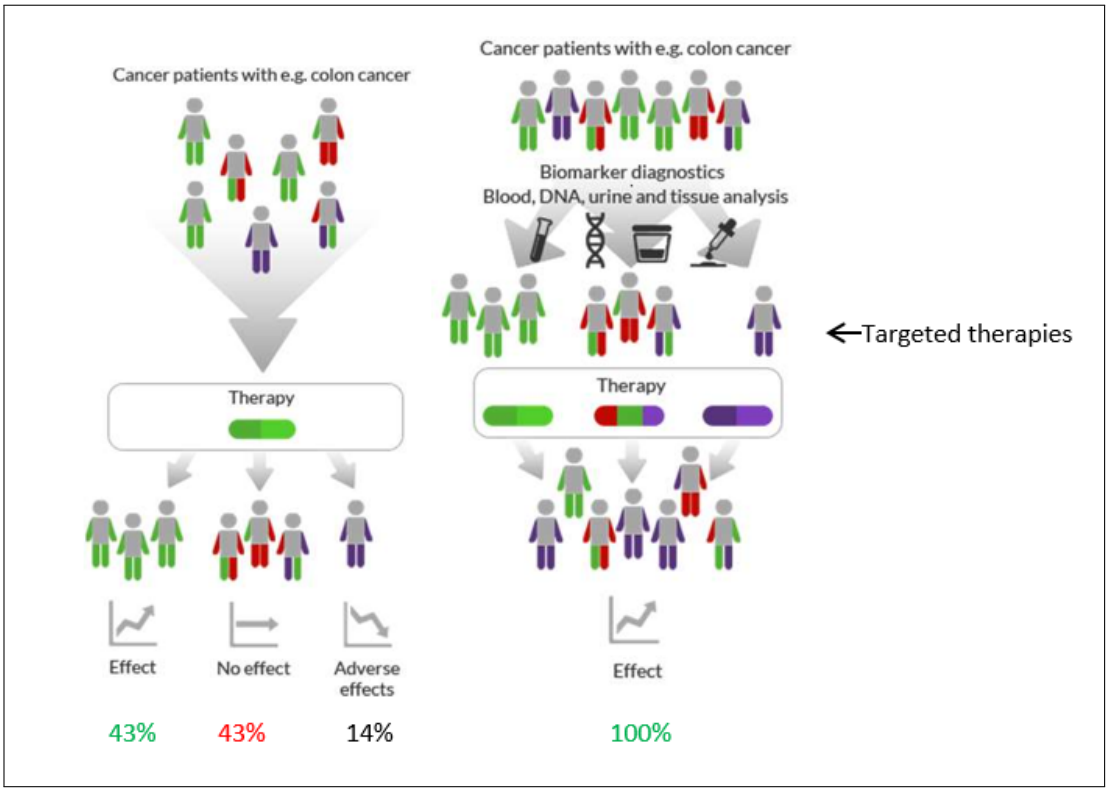


Figure 5.5: The repository contains over 32000 patient cases, including clinical data, treatment data, biopsy results, gene expression data, as well as a whole host of other information. There are about 21000 parameters associated with each of the patients, the majority of which is coming from gene expression data (>20,500), and the remaining are other pathological identifiers (>100). The objective is to improve overall survival days through root cause analysis and personalized medicine.

5.8.3 Model Training

A truly Lifelong Learning mechanism is developed by dividing the dataset as 50% for batch data at the initialization. The remaining data is ingested at a never-ending loop of 30 seconds sliding window interval. Training and test data volume ratio is kept 80% to 20% with one row relating to one patient records as provided in Tables 5.1 and 5.2. The proposed system aims to improve overall survival days by correlating the factors that influence survivability most. Data is first clustered by a hybrid model of batch and stream. The clustering step isolates the target data points, which then proceeds through a three-step process of sampling, feature selection, and dimension reduction to build a Decision Tree for reasoning and root cause analysis. Future elements are predicted by Decision Tree Classifier and Regressor for categorical and numerical fields.

5.8.4 Results with GDC Cancer Patient Data

This section presents the results for experiments with GDC cancer patient data starting with clustering results that pivot the focus area through outliers. Results from Random Forest drills down further to trace the possible consequences leading to a specific outcome. A comparative study is provided about the model performance against the standard machine learning algorithms (section 5.8.8) predicting categorical and numerical features. Following that, probable limitations of the model and alternative approaches are mentioned.

5.8.5 Experimental setup

The server setup is shown in Table 5.3. The database is Cassandra Datastax Enterprise Edition (DSE) with a replication factor of *three*. All nodes are installed with Cloudera Distribution of Hadoop (CDH) 6.1 and Apache Spark 2.3.1. Algorithms for streaming clustering and Decision tree and Random Decision Forest are developed with Scala 2.12.6. The total cluster resource is *100* cores, *375* GB memory, *25* TB of HDD. Each node is running 2 Spark executors in the Hadoop YARN cluster mode. Setting the number of executors per node is crucial to achieving optimal parallelism. Small executors decrease CPU and memory efficiency, and big executors

Table 5.3: AWS Instance Types

Instance Type	Instance Count	vCPUs	Memory	Instance Storage	EBS Optimized Bandwidth
m1.large	25	4	15 GB	1 TB	Moderate

may introduce a heavy GC overhead. Spark runs on the YARN cluster mode, where the Hadoop Resource Manager is the Spark Master and Node Manager is the executor. A Spark Driver running in YARN mode submits a job to the Resource Manager, and the Resource Manager delegates the task to an Application Master instance running in the cluster.

5.8.6 Clustering Results

The proposed stream-batch incremental learning and the streaming clustering methods are explained in Section 5.7 and Sub-section 5.7.1. This sub-section provides the evolution results for the clustering. Evolution used 50% of the data as historical records, and the remaining dataset is streamed at a rate of 10 records per 30 seconds sliding window interval to retrain the model iteratively. Visualizations are rendered by Splunk Machine Learning Toolkit [17]. Spark API based algorithms are implemented through the Python for Scientific Computing add-on in Splunkbase [18].

Data is clustered for patient profiling and detecting the outliers. Outliers are further analyzed with a Decision Tree for root cause analysis with an objective to improve the Overall Surviving (*XOS*) rate. Cluster charts for first biochemical recurrence and *Gleason Score* with *XOS* are shown in Figure 5.6 and Figure 5.7.

The clustering step also detects outliers through distance from the mean computed by the standard deviation (Figure 5.8 and 5.9).

The cluster plot reveals the gene expression level, along with the Gleason score, which has a strong correlation between being diagnosed with prostate cancer but no correlation to overall survival days. Figure 5.10 and Figure 5.11 show the overall survival by target gene TP53 when the Gleason score is categorized into three groups (6-7, 8, and 9-10).

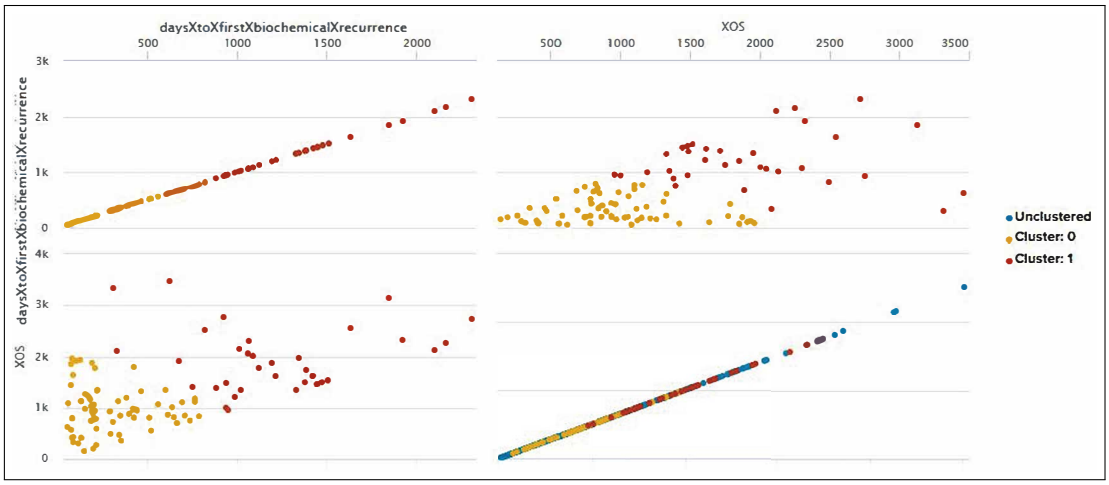


Figure 5.6: **Cluster chart for first biochemical recurrence (daysXtoXfirstXbiochemicalXrecurrence) with overall survival (*XOS*) in days in 3 clusters.** In this matrix representation of 4 charts, daysXtoXfirstXbiochemicalXrecurrence and *XOS* (along x-axis) are plotted with each other and with itself along the y-axis. Outliers are further analyzed using a Decision Tree for root cause analysis.

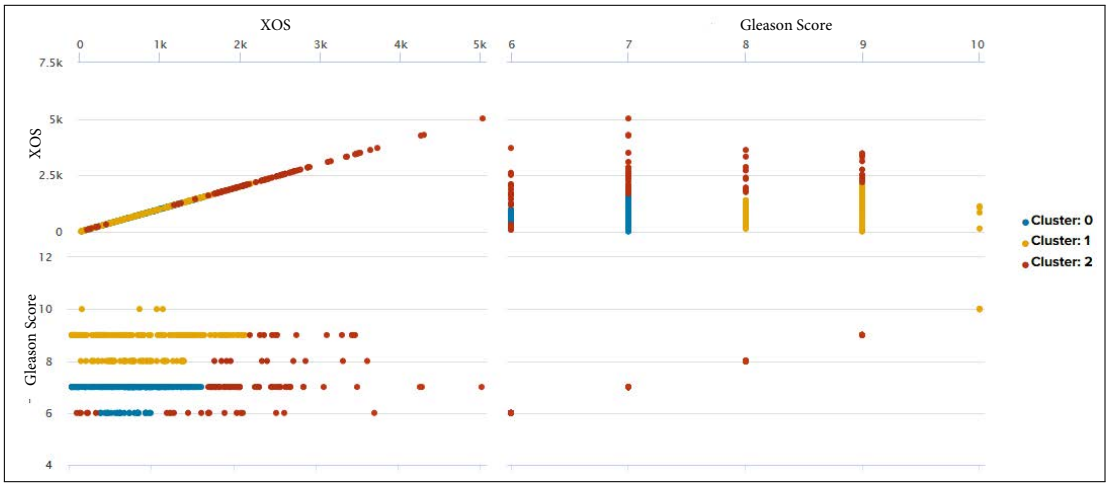


Figure 5.7: Cluster chart for *Gleason Score* (gleasonXscore) with overall survival (*XOS*) in days in 3 clusters. In this matrix representation of 4 charts, gleasonXscore and *XOS* along with the x-axis are plotted against each other and with itself along with the y-axis. Outliers are further analyzed using a Decision Tree for root cause analysis.

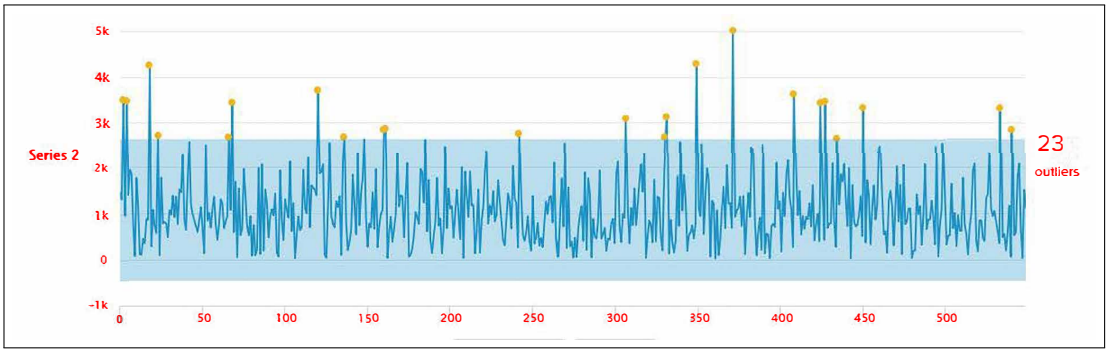


Figure 5.8: Outliers chart for anomalous Overall Survival (*XOS*) data. Total 23 outliers are detected (yellow dots) which are further analyzed with Decision Tree for Root Cause Analysis (RCA). Anomalous values are detected through distance from the mean as shown in Figure 5.9.

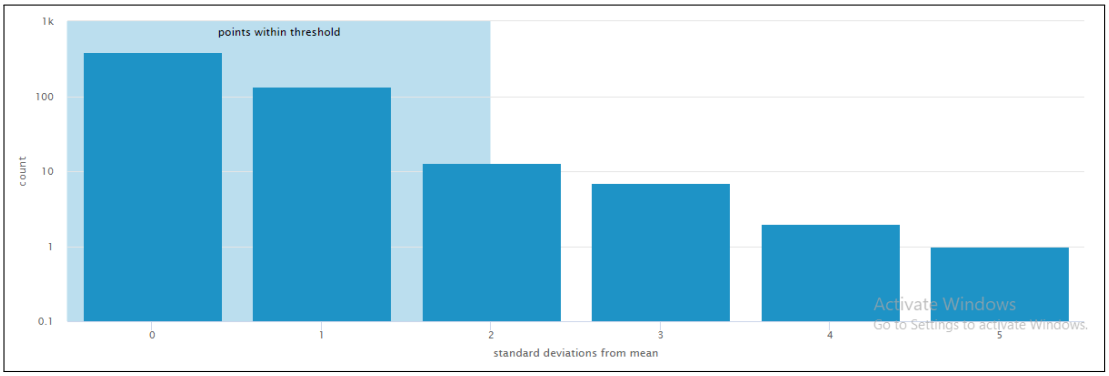


Figure 5.9: Anomalous values are detected as distance from the mean computed by standard deviation. The shaded area illustrates the data points within the allowable threshold.

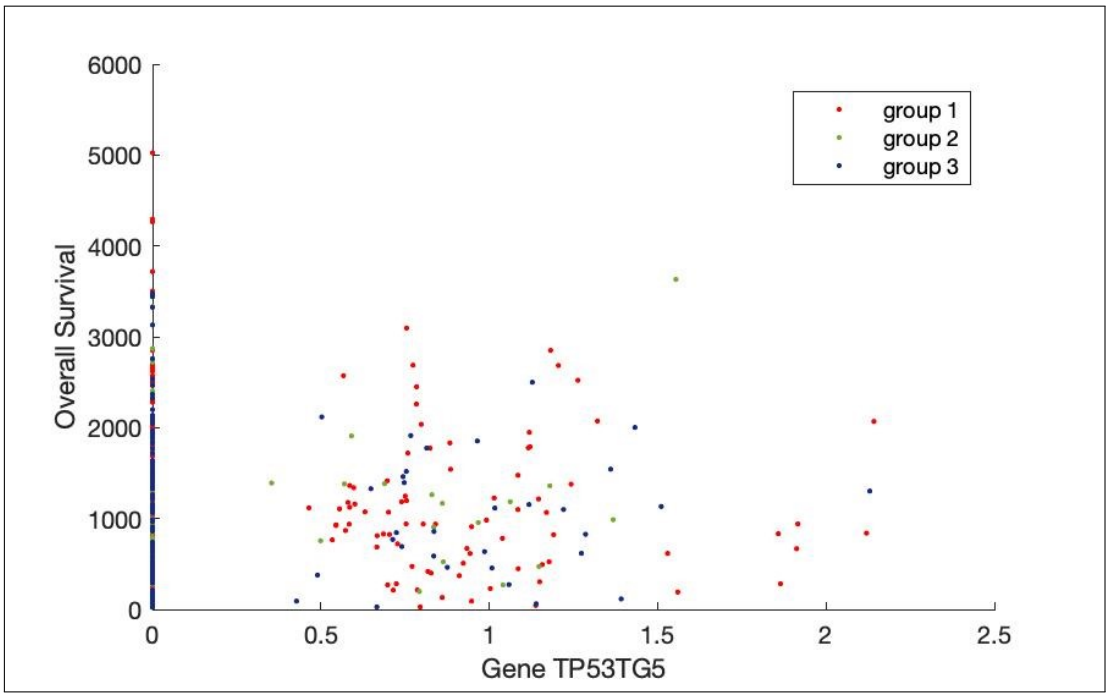


Figure 5.10: Overall survival by cancer-causing gene TP53TG5 along with Gleason scores clustered into three groups (6-7, 8, and 9-10 as group 1 to 3). The data is clustered by a proposed hybrid streaming clustering step discussed in the section 5.7.1. The plot shows a strong possibility of prostate cancer with a *gene TP53TG5* expression level of 0.5 to 1.5 and a Gleason score between 6 to 7.

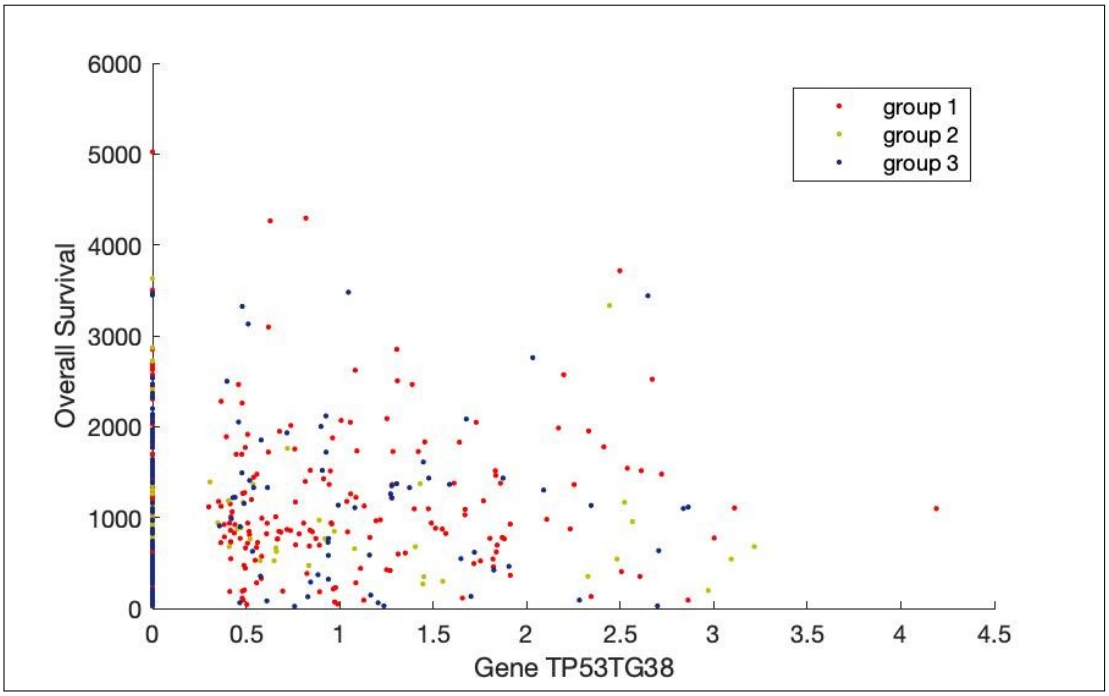


Figure 5.11: Overall survival by cancer-causing gene TP53TG35 along with Gleason scores clustered into 3 groups (6-7, 8, and 9-10). The data is clustered by the proposed hybrid streaming clustering step discussed in Section 5.7.1. Prostate cancer patients have a much higher probability of TP53TG35 gene expression level between 0.5 to 3 with a Gleason score between 6 to 7.

To remark about the outcomes, it is evident that the clustering methods provide an efficient and actionable grouping to detect the outlier for the patient profiles. Subsequently, anomalous behaviour is correctly summarized in the outlier chart. The efficiency of the process is improved over time with the incremental Lifelong Learning approach. Figure 5.9 shows an explanation for outlier detection.

5.8.7 Random Decision Forest results

The Random Decision Forest (RDF) Regressor graph explains the possible consequences leading to survival days. The proposed method for constructing the RDF is described in Section 5.7 and Sub-sections 5.7.2 and 5.7.3. Figure 5.12 shows the Decision Tree splitting process on the portion of the tree. It appears from the tree that there are significantly contributing genotypes, phenotypes, and other clinical parameters. For instance, *days to first biochemical recurrence* is an influencing attribute clearly dividing the data pool. A similar conclusion can be derived for genes like *KFF6* or *LRP2*.

Figure 5.13 compares between influent human genes contributing towards lower Overall Survival (*XOS*) in days. A Random Decision Forest classifier detects the commonly present genes among prostate cancer patients, and the chart further breaks down the results by ranking among the available gene pool.

Random Decision Forest provides reasoning through the Decision Tree graph for the root cause analysis on every anomalous cluster detected by the previous clustering step. Observe, the overall survival rate is positively influenced if Gleason score <7, if the sample type is not the primary tumour, the patient did not receive radiation therapy and days to biochemical recurrence >200 as identified by the red lines in Figure 5.15.

The Decision Tree shown in Figure 5.15 accurately indicates the Gleason score as a contributing factor for the overall survival days (see red lines). Figure 5.16 by the Random Decision Forest algorithm further proves a Gleason score between 6 to 9 has a much lower survival rate of fewer than 1000 days. Finally, a Kaplan-Meier estimator plot [50] [195] is presented based on the Ran-

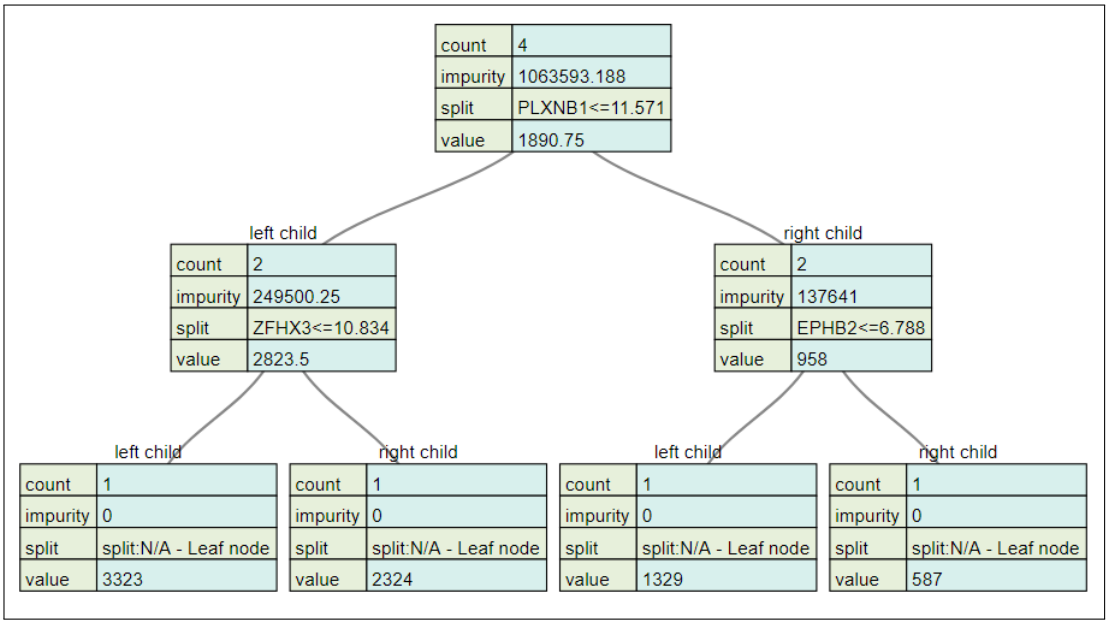


Figure 5.12: **The Decision Tree split process.** Part of the tree is displayed here. The information gain for each feature variable is computed. The feature with the highest information gain is chosen as the root and the splitting node. In each node, information gain along with the impurity level is measured after each split. The process is iterated until the leaf nodes are generated.

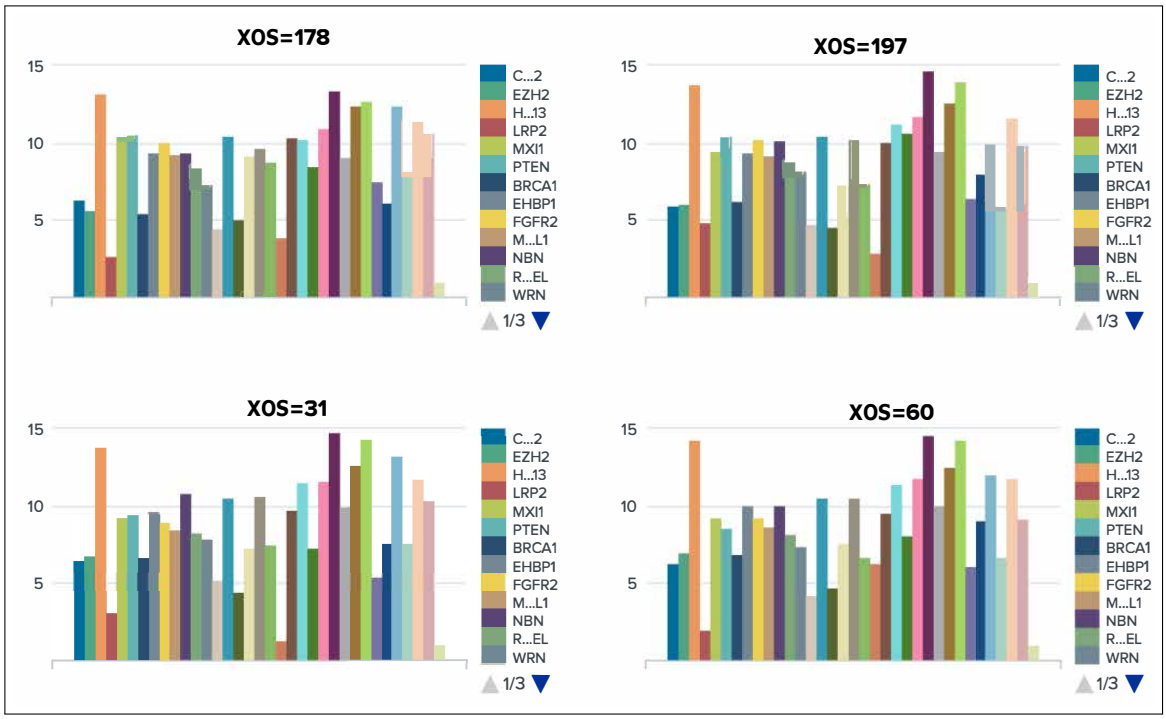


Figure 5.13: Overall Survival (*XOS*) in days (along with the x-axis) is charted against the influent genes. While a Random Decision Forest classifier detects the commonly present genes among prostate cancer patients, this chart identifies the top human genes influencing *XOS* values as follows: HOXB13, MSMB, and CDH1.

Predicted actual	Predicted 7	Predicted 8	Predicted 9	Predicted na
7	5528 (92%)	472 (8%)	0 (0%)	0 (0%)
8	1015 (22%)	3562 (77%)	3 (0.06%)	0 (0%)
9	1 (0.03%)	1015 (33%)	2045 (66.8%)	0 (0%)
na	0 (0%)	0 (0%)	1 (100%)	0 (0%)

Figure 5.14: **Confusion Matrix for MALA's categorical forecasting.** Along the diagonal cells are counts for the correct predictions placed, counts for incorrect predictions are placed in the rest of the cells.

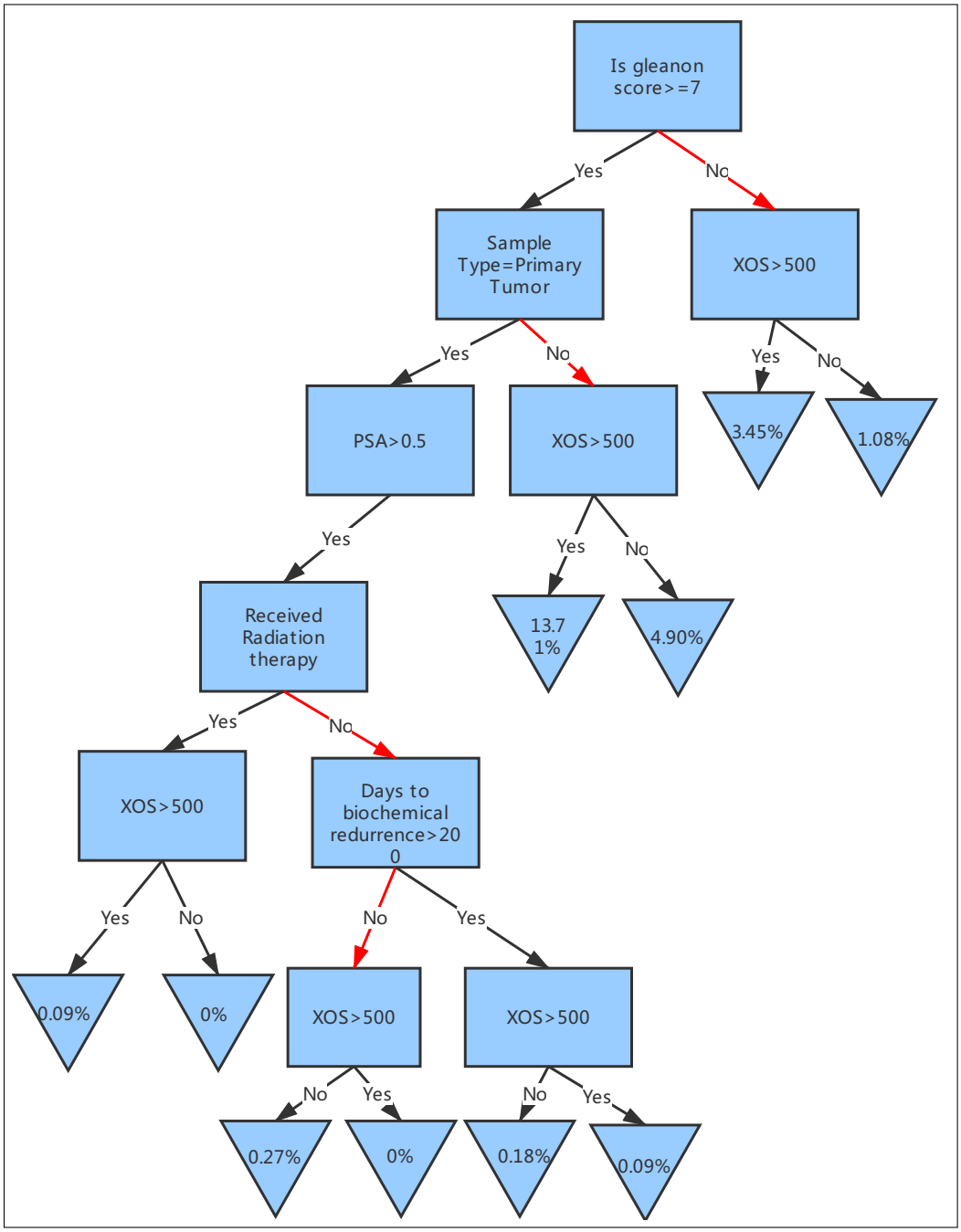


Figure 5.15: **Random Decision Forest drills down for the anomalous cluster.** The Decision Tree provides reasoning with root cause analysis. The red lines in the Decision Tree show the positive influence of the Gleason score (<7), sample type (not a primary tumor), no radiation therapy and days to biochemical recurrence (>200) on overall survival rate (*XOS*).

dom Decision Forest algorithm. The estimator of survival is given by Equation 5.6.

$$\hat{S}(t) = \prod_{i:t_i \leq t} (1 - \frac{d_i}{n_i}) \tag{5.6}$$

where d_i and n_i are the number of deaths and survival at time t_1. See Figure 5.17.

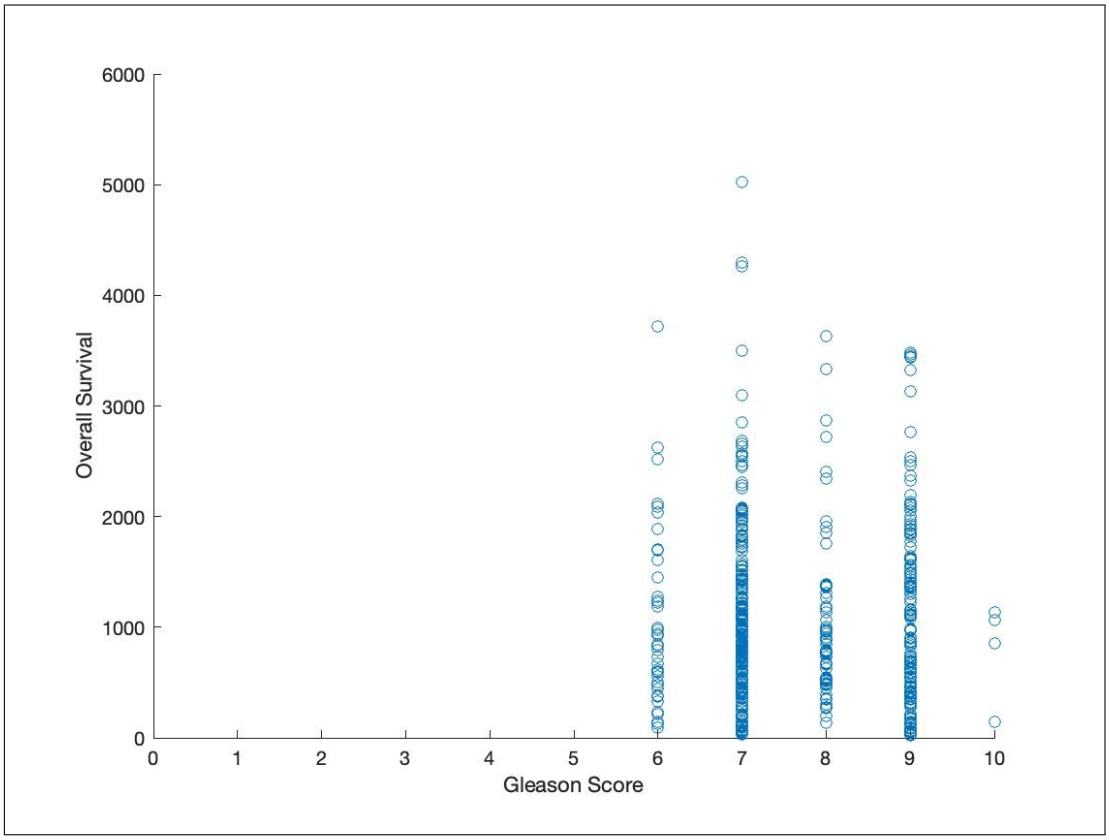


Figure 5.16: Gleason score with overall survival days.

5.8.8 Model Comparisons

Several experiments are conducted comparing the proposed MALA with a Decision Tree Regressor, Kernel Ridge, Elastic Net, Ridge, Linear Regression, and Lasso models for numeric field predictions. Models are compared in terms of R^2 statistics and RMSE. In general, higher R^2 and

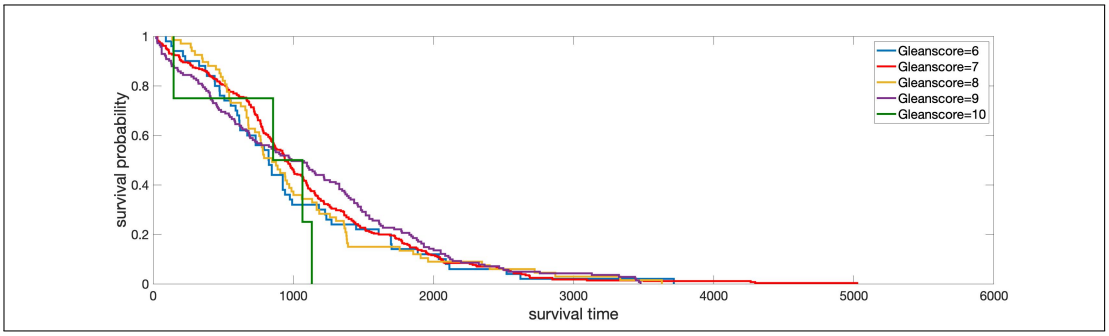


Figure 5.17: Kaplan-Meier estimator plot for the Gleason score.

lower RMSE signify the model fitting the data better. Results are presented in Table 5.4. Model prediction accuracy for categorical fields is presented in Table 5.5. In categorical forecasting, the proposed MALA is compared with GaussianNB, Bernoulie NB, Decision Tree Classifier, SVM, and Logistic Regression in terms of Precision, Recall, Accuracy, and F1 Score. A good model is supposed to exhibit higher values for all comparison parameters. The split for the training and test dataset was 50% for both experiments. Since the MALA uses both batch and stream data to fit its model, 50% of the training dataset is used to train at batch setting, and the remaining is used at streaming setting.

See the reports for actual vs predicted Line Chart and Scatter Chart (Figure 5.20), Residuals Line Chart, and Residuals Error Histogram (Figure 5.21) for comparative models. Lower values in the Residuals Line Chart and Residuals Error Histogram should prove good model predictions.

The classification accuracy for MALA is 4% greater than SVM, 29% greater than Bernoulie NB, and more than 30% greater than all other algorithms. In terms of regression RMSE, the proposed model is 4% better than the Elastic Net, 37% better than the Decision Tree Regressor and more than 40% better than the rest of the algorithms. Furthermore, due to the lifelong incremental learning capabilities, the proposed model significantly improves over time without adding latency in training time. Therefore, compared with comparative models, MALA improves classification and regression accuracy significantly for multiple scales and parameters.

Table 5.4: Model Prediction Accuracy for Numeric Fields

Algorithm	R^2 **Statistics**	**RMSE**
Decision Tree Regressor	-0.9600	1149.84
Kernel Ridge	-4.4322	1678.87
Elastic Net	-0.2219	875.24
Ridge	-12.8802	3032.13
Linear Regression	-8.1000	2213.07
Lasso	-13.8496	2768.54
Proposed MALA[1]	*-0.3840*	*839.14*

The proposed model is 4% better than the Elastic Net, 37% better than the Decision Tree Regressor and more than 40% better than the rest of the algorithms.[1]

Table 5.5: Model Prediction Accuracy for Categorical Fields

Algorithm	**Precision**	**Recall**	**Accuracy**	**F1 Score**
GaussianNB	0.10	0.17	0.17	0.13
Bernoulie NB	0.36	0.52	0.52	0.43
Decision Tree Classifier	0.45	0.50	0.50	0.47
SVM	**0.76**	0.71	0.71	0.73
Logistic Regression	0.34	0.48	0.48	0.40
Proposed MALA[2]	*0.74*	**0.74**	*0.74*	*0.74*

MALA is 4% greater than SVM, 29% greater that Bernoulie NB and more than 30% greater than all other algorithms. [2]

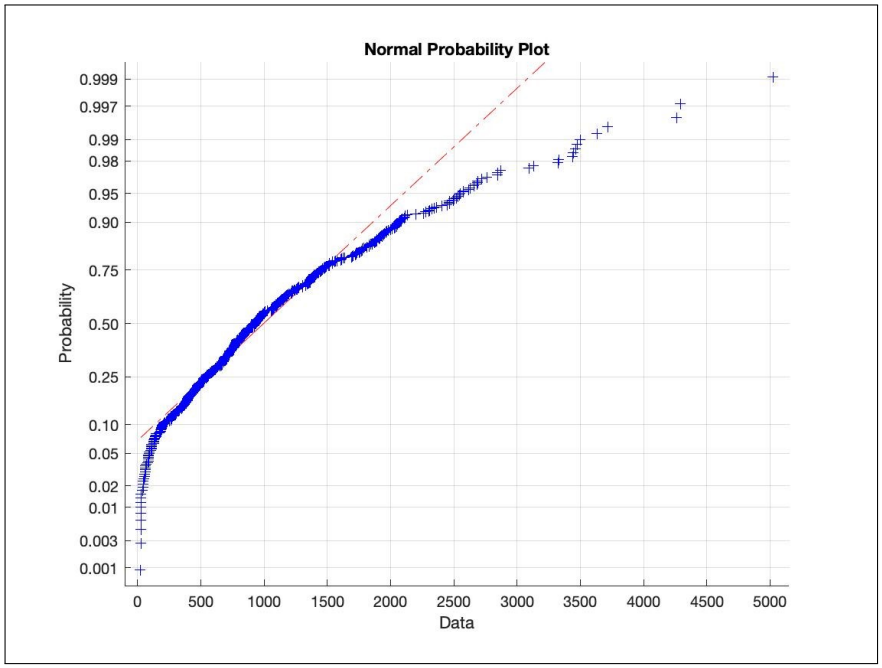


Figure 5.18: Lilliefors test.

5.8.9 Results Analysis

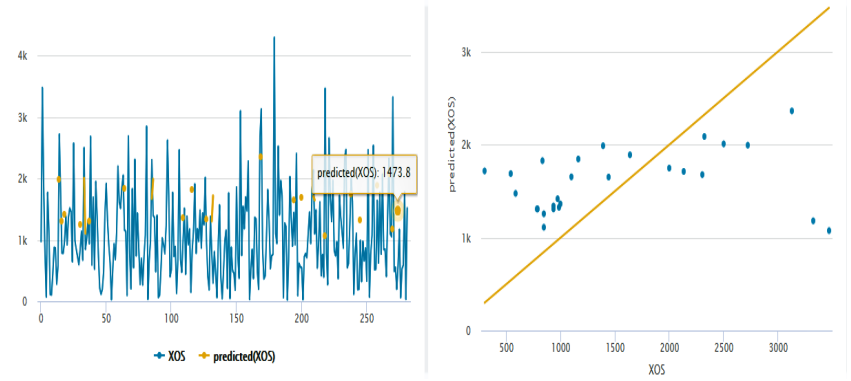
The MALA provides reasoning through dynamic root cause analysis. The framework's success largely depends on the accuracy of reasoning illustrated by the decision graph. For instance, results highlight, the Gleason scores categorized into three groups (6-7, 8 and 9-10) have a major influence on the overall Survivability (*XOS*). To verify the claim, the probability distribution plot of *XOS* can identify if *XOS* follows a normal distribution through the Lilliefors test [205] [33] given by:

h = lillietest(x)

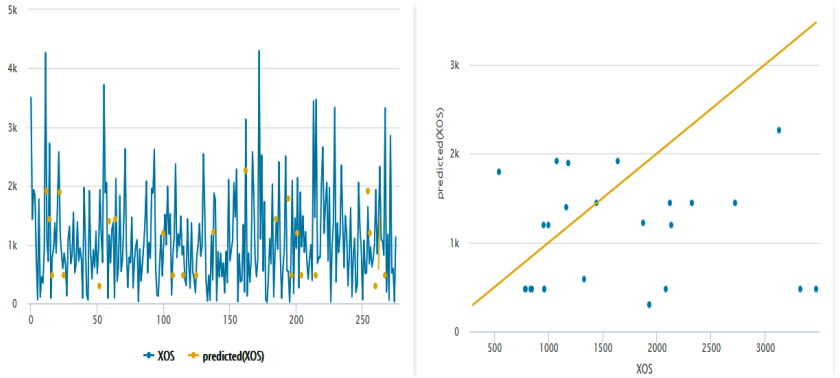
The value of *h* is *one* if the test rejects the null hypothesis at the 5% significance level, and *zero* otherwise.

The Lilliefors test rejects the null hypothesis at the 5% significance level. Therefore, *XOS* does not satisfy normal distribution. See Figure 5.18.

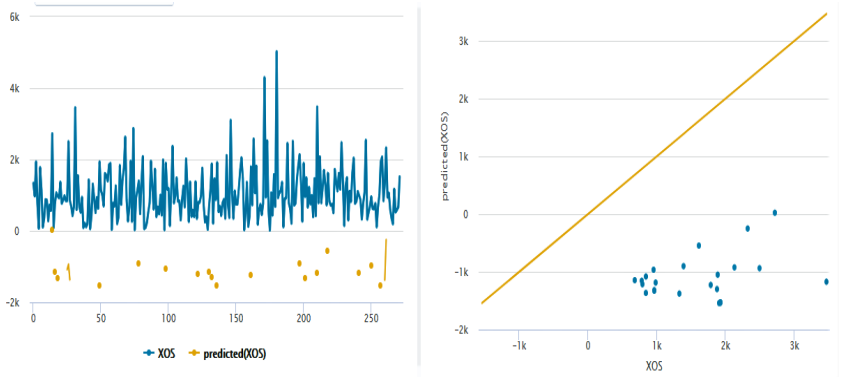
Given *XOS* does not satisfy normal distribution, other methods are used to find if the Gleason



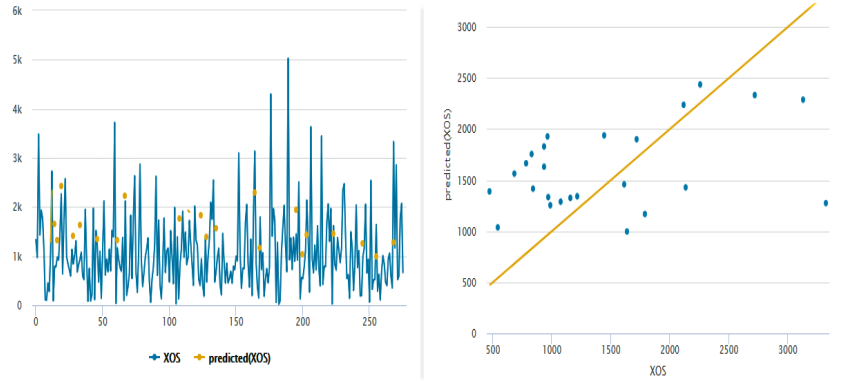
(a) Proposed MALA



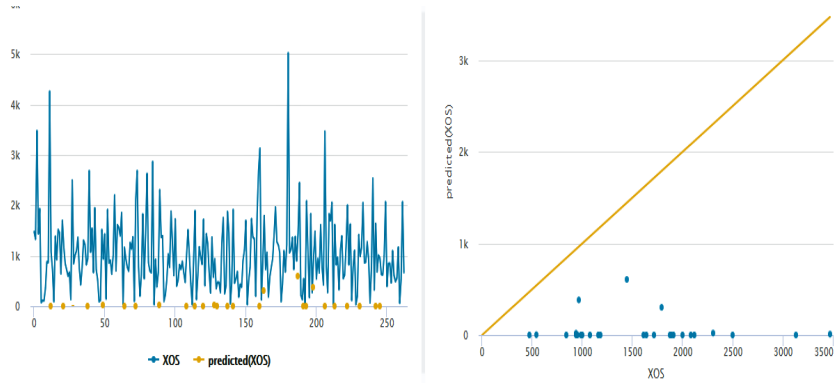
(b) Decision Tree Regressor



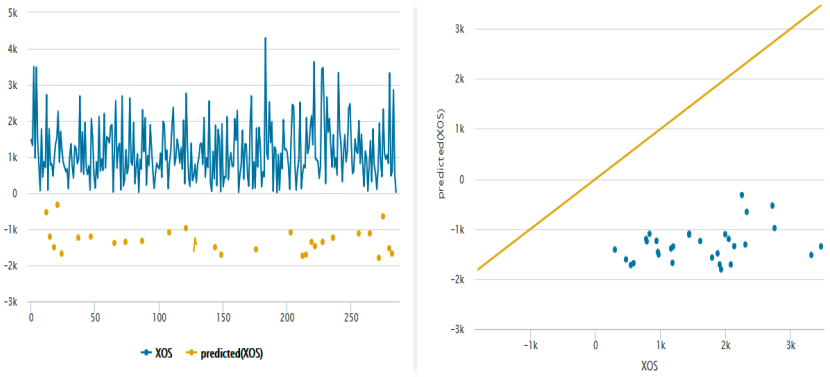
(c) Lasso



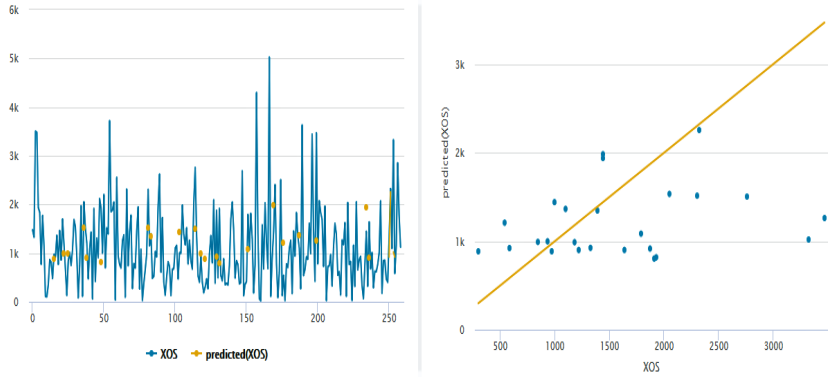
(d) Linear Regression



(a) Kernel Ridge

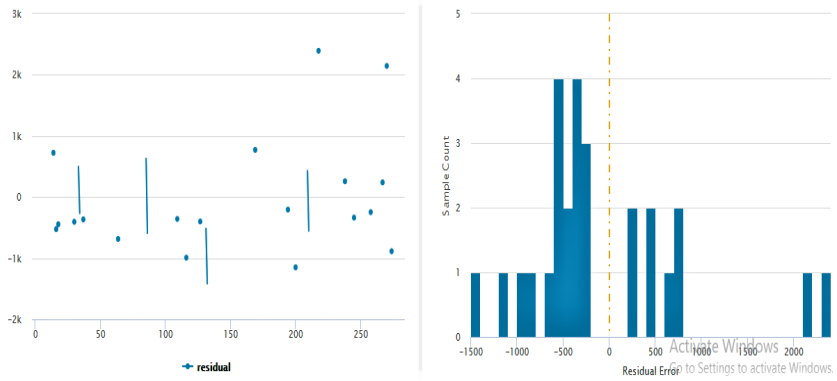


(b) Ridge

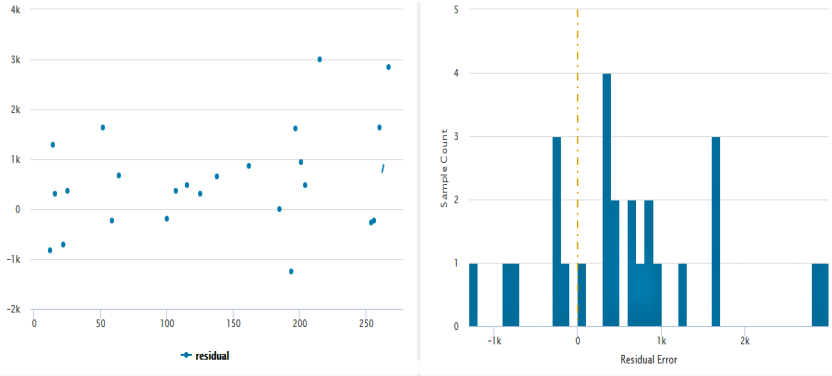


(c) Elastic Net

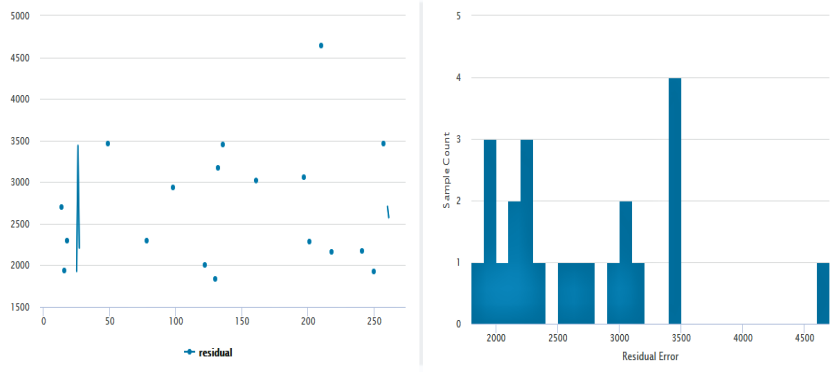
Figure 5.20: **Performance comparison of different machine learning models.** Actual vs predicted Line Chart and Scatter Chart. Blue and yellow lines or dots show the actual and predicted values. The proposed MALA shows the higher accuracy than remaining models.



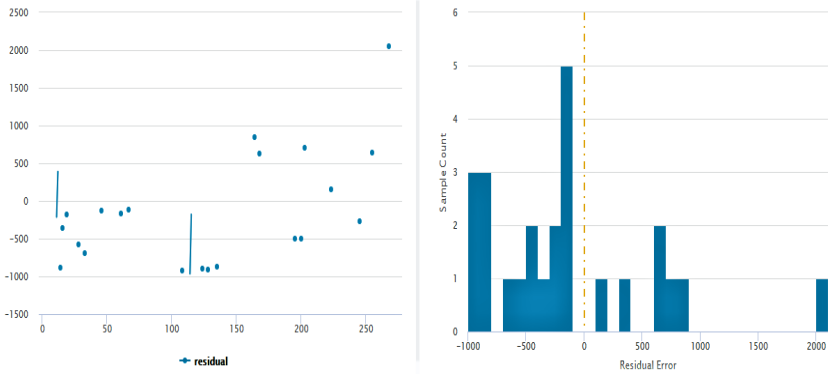
(d) Proposed MALA



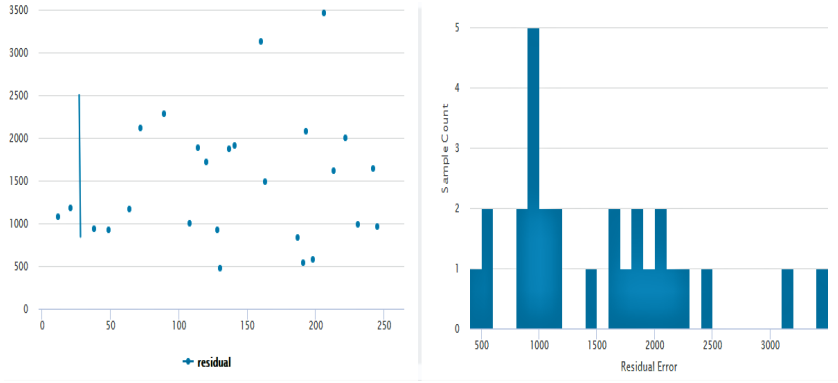
(e) Decision Tree Regressor



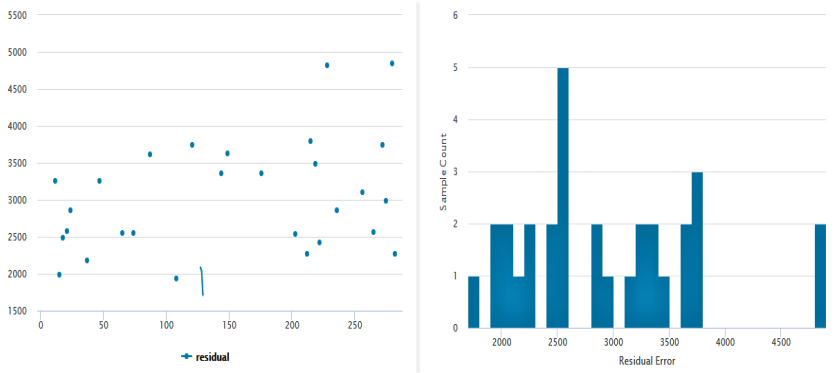
(f) Lasso



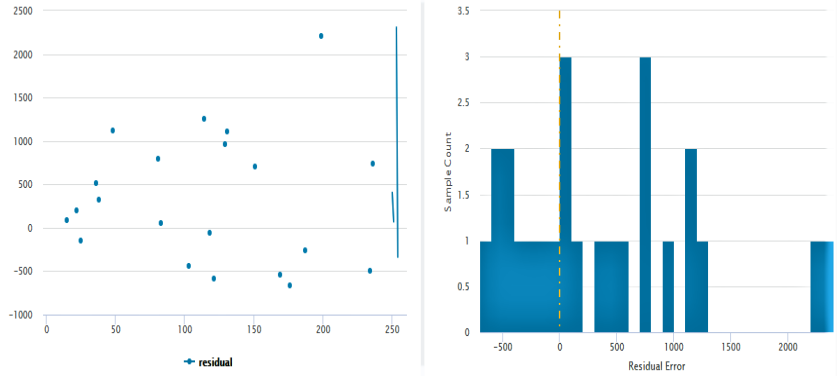
(g) Linear Regression



(a) Kernel Ridge



(b) Ridge



(c) Elastic Net

Figure 5.21: **Performance comparison of different models.** Residuals Line Chart and Residuals Histogram.

score influences the overall survival. Hence, *Two-sample Kolmogorov-Smirnov test* [152] [168] is chosen to validate if the Gleason score has an influence on the overall survival rate. Kolmogorov-Smirnov test determines if two of the one-dimensional probability distribution curves differ. The null hypothesis is, Gleason score groups (6-7, 8 and 9-10) *does not* have any influence on the overall survivability. The objective is to deny the null hypothesis and prove the model's reasoning favourably. We call the Two-sample Kolmogorov-Smirnov test:

$$[h,p] = kstest2(XOS(10 < Gleason\ Score < 6), \\ XOS(group[1/2/3]), \alpha, 0.1) \tag{5.7}$$

α is the Significance level, kept at 0.05 (default). The Test returns an h and a *p-value*. If h=1 we can reject the hypothesis (Equation 5.7), which means the two vectors *XOS* (10<*Gleason Score*<6) and *XOS* (group[1/2/3]) have the different distributions as the test is sensitive to location and shape of the cumulative empirical distribution functions of the two samples. A small p-value (typically ≤ 0.1) disproves the null hypothesis. The result is shown Table 5.6 at 10% significance level.

Table 5.6: Student's t-test

Returned Parameters	**Group 1**	**Group 2**	**Group 3**
h-value	1	1	1
p-value	0.09	0.10	0.10

The value of h is a test decision for the null hypothesis to verify if the two groups of data are from the same continuous distribution, i.e. not significantly different. The value of h is *one*, if the test rejects the null hypothesis at the 10% significance level, and *zero* otherwise. A positive correlation is reported between *Gleason score* and *overall survivability* from Table 5.6, with h=1 and $p \leq \alpha$ (threshold) for all 3 groups. A small *p-value* provides evidence against the null hypothesis with a significant distance between cumulative empirical distribution functions for two samples. The populations' distribution of curves may differ in the shape of the distribution, variability, or median. However, in the test, the correlation seems to be tenuous with relatively bigger p values.

Therefore, the results conclude that the different Gleason score groups (6-7, 8 and 9-10) *does* have an influence on the overall survivability.

5.8.10 Limitations of the proposed model

The proposed framework has a limitation of managing the identical code base for batch and stream layer, which yields the same results. The code redundancy may lead to a typical sync problem since changes in one layer (batch or stream) requires to reflect the changes to another layer. Also, when comparing accuracy, the model would be no superior to the batch only model when trained on the same data volume. Nevertheless, the Incremental Lifelong Learning framework gradually improves the prediction accuracy over time and with additional data. One of the shortcomings of the *Kolmogorov-Smirnov test* is that the test is overly responsive with every possible kind of deviation between two sample distribution functions. As an alternative to Kolmogorov-Smirnov tests, *Area Under the Curve (AUC)* [66] [198] can also be adapted to measure the efficiency of the proposed model in distinguishing different *Gleason Score* groups.

5.9 Sentiment Polarity through Lifelong Learning

This section applies the Lifelong Learning model to analyzing *Sentiment Polarity*. Computing *sentiment polarity* is a two-step process. The *first step* is the initial learning stage, building on top of a large volume of the historical dataset at a *batch setting*. The second step is a *self-study* stage, where a previously trained model applies itself for self-learning and prediction at a *streaming* setting. The section extends the proposed methods described in Sections 5.6 and 5.7, followed by a case study implemented through crawled records from the Amazon dataset containing user reviews from 20 domains (product categories) with each domain having about 1,000 reviews. Review distribution is imbalanced with a mix of positive and negative reviews.

Although deep learning is widely applied in sentiment classification, it still could not leverage past acquired knowledge efficiently. The inherent complexity of neural networks limits the

research to defining and extracting knowledge from unstructured data. This research uses the Naïve Bayes since knowledge can be presented in terms of probability. The approach consists of a knowledge transfer and a knowledge validation method. Although Lifelong Sentiment Classification (LSC) [64] introduces a continuous learning approach, the method only aims to improve classification accuracy. The LSC is still strictly under the setting of supervised learning and is unable to deliver explicit knowledge to guide the future learning process. In that context, this work presents a Lifelong Learning paradigm to overcome the constraints in sentiment classification under unsupervised learning settings with previously harvested knowledge.

To achieve the goal of *strong AI* through Lifelong Learning, this case study focuses on *sentiment polarity* of *words*, developing an algorithm to define how each word in a corpus influences the sentiment of the entire corpus of documents by knowledge reuse. If a model can achieve this learning goal, the algorithms can solve new tasks *without further training*. Chen [64] proposed a model with a similar learning objective but supervised learning still was required. Guangyi [159] extended the work of [64] with a neural network-based approach. However, supervised learning is still necessary under their settings, and a large volume of labelled data is required. Hence, this work aims to decrease the usage of labelled data while maintaining the performance level in terms of accuracy.

5.9.1 The Naïve Bayesian Text Classification

Sentiment polarity is computed by calculating the *probability* of each word appearing within a positive or negative context in a sentence or document. Suppose *words* with a document have a high probability of having sentiment polarity. In that case, the *document* also can have a higher probability of sentiment probability based on the Naïve Bayesian (NB) classifier. Therefore, determining the words with *polarity* is the key to predict the overall sentiment in a corpus.

The Naïve Bayesian (NB) classifier [170] computes the *probability of sentiment* for each word w in a document d and then predicts the sentiment polarity (positive or negative). The Equation 5.8 refers to the probability of a word having sentiment polarity given the document classes (positive

or negative):

$$P(w|c_j) = \frac{\lambda + N_{c_j,w}}{\lambda |V| + \sum_{v=1}^{V} N_{c_j,v}} \tag{5.8}$$

Where c_j is either positive (+) or negative (-) sentiment polarity. $N_{c_j,w}$ is the frequency of a word w in documents of class c_j . |V| is the size of vocabulary V and $\lambda (0 \leqslant \lambda \leqslant 1)$ is used for smoothing (set as *1* for Laplace smoothing).

Given a document, we can calculate the sentiment probability for different classes (positive or negative) by:

$$P(c_j|d_i) = \frac{P(c_j)\prod_{w\in d_i} P(w|c_j)^{n_w,d_i}}{\sum_{r=1}^{C} P(c_r)\prod_{w\in d_i} P(w|c_r)^{n_w,d_i}} \tag{5.9}$$

Where d_i is the given document, n_w, d_i is the frequency of a word appears in this document.

To predict the class of a document, we only need to calculate $P(c_+|d_i) - P(c_-|d_i)$. If the polarity score is lager than 0, the document is predicted with positive polarity, as given by the following equation:

$$P(c_+|d_i) - P(c_+|d_i) = \frac{P(c_+)\prod_{w\in d_i} P(w|c_+)^{n_w,d_i}}{\sum_{r=1}^{C} P(c_r)\prod_{w\in d_i} P(w|c_r)^{n_w,d_i}} - \frac{P(c_-)\prod_{w\in d_i} P(w|c_-)^{n_w,d_i}}{\sum_{r=1}^{C} P(c_r)\prod_{w\in d_i} P(w|c_r)^{n_w,d_i}} \tag{5.10}$$

We only need to know whether $P(c_+|d_i) - P(c_-|d_i)$ is lager than *0*, Equation 5.10 is simplified to Equation 5.11:

$$P(c_+|d_i) - P(c_+|d_i) = P(c_+)\prod_{w\in d_i} P(w|c_+)^{n_w,d_i} - P(c_-)\prod_{w\in d_i} P(w|c_-)^{n_w,d_i} \tag{5.11}$$

5.9.2 Discovering Words with Sentiment Polarity

At an ideal setting, if we know the $P(c_+)$, $P(c_-)$ and $P(w|c_j)$ for all of the words, we can predict the sentiment polarity for the entire documents. The LSC came up with a possible solution to calculate $P(w|c_j)$, but it uses *all of the words* in the corpus leading to overfitting. Given that, some words do not have sentimental polarity like *"a", "one"* and, etc. While some words always have polarity like "*good", "hate", "excellent"* and so on. In addition, some words only have sentiment polarity within a specific context. For example, *"tough"* in reviews of the diamond indicates that the diamond has a good quality (positive sentiment). At the same time, it means hard to chew in the domain of food (positive Sentiment). We need to find the words with sentiment polarity in all occurrences and separate words that only show polarity for a specific domain.

5.9.3 Lifelong Semi-supervised Learning for Sentiment Classification

A new learning method is proposed in two stages:

1. *Initial Learning Stage (at the batch mode):* Explore a basic set of sentiment words. Thereafter, the model should be able to classify a new domain with good performance.

2. *Self-study Stage (at the streaming mode):* Use the knowledge accumulated from the initial stage to handle new domains, fine-tune, and consolidate the knowledge generated from the initial learning stage.

The method is explained in detail as follows:

Initial Learning Stage

At this stage, the model is trained using Naïve Bayes classifier on a large pool of *historic data* under the *batch mode* computing sentiment polarity of words. The process finds the words with *sentiment* in each domain (Amazon.com product category in the current case study), determining the polarity of a word. A word w can have positive polarity, if $P(+|w) >> P(-|w)$ or $P(+|w) >>$

$P(+)$. Polarity is represented by the equation: $O(w) = P(+|w)/P(+)$. According to the Bayesian equation, $P(+|w)/P(+) = P(w|+)/P(w)$.

At the initial learning stage, a set of words are filtered based on their influence on the overall sentiment classification for the document. We can calculate $P(w|+)$ and $P(w|-)$ for each word. The polarity score is defined by $O(w) = P(w|+)/P(w)$. Words or symbols are sorted by the polarity $O(w)$, and then a certain proportion of words or symbols are selected from the entire set of words. From Table 5.7, we can observe that using no less than a 30% data sample obtains the best result. It is apparent that most words and symbols do not have obvious sentiment orientation. Hence, the process only keeps 30% of words for the Naïve Bayes classifier and even get a better *f1* score. Although the performance decreases with only a single domain, better *overall* performance can be achieved by considering only the words consisting of sentiment.

Number of domains for the initial learning. In practice, all labelled data are required for training, but an efficient model can determine the amount of labelled data that meets the minimum performance requirement. For the sentiment classification task, analysis based on a single domain can be insufficient. Based on the experimental observations, the initial learning stage needs at least *two* domains and can achieve significantly better performance with three domains. An increasing number of domains further may not significantly influence performance. As per the experimental evolution, *three domains* are sufficient for this task. Adding a new domain can continue until the performance tends to remain unchanged after a new domain addition.

Self-study Stage

The task is to detect the words with sentiment polarity at the next stage, using these words to predict the new domains and assign the pseudo-labels to them. With the pseudo labels, the model can discover new words having sentiment polarity. Following are the steps for the self-study:

1. Use the sentiment words accumulated from the current domain to predict sentiment labels (positive or negative) on a new domain, then assign the prediction results as the *pseudo labels*. Prediction at this step uses Valence Aware Dictionary for Sentiment Reasoning

(VADER). VADER does not require any training data but is developed using a rule-based, human-constructed, gold standard sentiment dictionary [114]. For more detail about sentiment analysis using VADER see Section 7.4.3.

2. Reuse pseudo labels and actual reviews of the new domain as new training data to run the Naïve Bayes classifier described in Section 5.9.1.

3. Update the knowledge base with the latest trained model using Naïve Bayes classifier.

The self-study is a re-learning stage under the *supervised* and *stream* setting with the *pseudo labels* from VADER. Once the training model is built at the *initial learning stage* under the batch settings, VADER predicts sentiment on a new domain under the stream settings and prediction outcomes (positive or negative sentiments) are considered as pseudo labels for the next step. After that, the Naïve Bayes model regards the pseudo labels as the real labels and continue training on a new domain. Therefore, with this method, the self-study learning stage can learn new domains efficiently without further labelled data.

Table 5.8 is the *F1* score of *three* models on *17* domains. The first three domains are used for the initial learning stage. *Macro-F1 score* is used since the datasets are imbalanced, and it can improve the performance of the smaller classes. The model (Semi-Unsupervised Learning, SU-LML for short) is compared with the Naïve Bayes model. The SU-LML model is trained only on the first three source domains, NB-S is trained on all domains with labels, and NB-T is trained on all domains by 5-fold cross-validation. We can observe that the proposed approach is significantly better than the other two approaches. The method even outperforms the NB-T, typically a supervised learning method. Figure 5.22 shows the results more precisely. The comparisons to the LSC and neural-based Lifelong Learning [159] could not be presented at the current circumstances since their approaches are completely based on supervised learning and also due to the unavailability of the codebase in the public domain.

F1 Score / Percentage / Datasets	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%
AlarmClock	0.8082	0.8082	0.8082	0.8082	0.8082	0.8082	0.8082	**0.8082**	0.274	0.2333
Baby	0.6564	0.6564	0.6564	0.6564	0.6564	0.6564	0.6564	**0.6564**	0.1759	0.1408
Bag	0.6811	0.6811	0.6811	0.6811	0.6811	0.6811	0.6811	**0.6811**	0.3559	0.1056
CableModem	0.6064	0.6064	0.6064	0.6064	0.6064	0.6064	0.6064	**0.6064**	0.2195	0.1105
Dumbbell	0.6346	0.6346	0.6346	0.6346	0.6346	0.6346	**0.6346**	0.6602	0.1589	0.1383
Flashlight	0.5876	0.5876	0.5876	0.5876	0.5876	0.5876	0.5876	**0.5921**	0.3278	0.1036
Gloves	0.6131	0.6131	0.6131	0.6131	0.6131	0.6131	0.6131	**0.6131**	0.3205	0.1206
GPS	0.6814	0.6814	0.6814	0.6814	0.6814	0.6814	0.6814	**0.6814**	0.2838	0.1629
GraphicsCard	0.5775	0.5775	0.5775	0.5775	0.5775	0.5775	0.5775	**0.5775**	0.2776	0.1271
Headphone	0.6578	0.6578	0.6578	0.6578	0.6578	0.6578	0.6578	**0.6578**	0.268	0.1745
HomeTheaterSystem	0.8394	0.8394	0.8394	0.8394	0.8394	0.8394	0.8394	**0.8394**	0.2404	0.2238
Jewelry	0.604	0.604	0.604	0.604	0.604	0.604	0.604	**0.604**	0.3371	0.1088
Keyboard	0.653	0.653	0.653	0.653	0.653	0.653	0.653	**0.653**	0.2117	0.1841
MagazineSubscriptions	0.8042	0.8042	0.8042	0.8042	0.8042	0.8042	0.8042	0.8042	**0.8049**	0.2115
MoviesTV	0.5843	0.5843	0.5843	0.5843	0.5843	0.5843	0.5843	0.5843	**0.606**	0.0976
Projector	0.7387	0.7387	0.7387	0.7387	0.7387	0.7387	0.7387	**0.7387**	0.1814	0.168
RiceCooker	0.7656	0.7656	0.7656	0.7656	0.7656	0.7656	**0.7656**	0.7739	0.1683	0.1566
Sandal	0.5987	0.5987	0.5987	0.5987	0.5987	0.5987	0.5987	**0.5987**	0.3501	0.1077
Vacuum	0.7362	0.7362	0.7362	0.7362	0.7362	0.7362	0.7362	**0.7362**	0.2155	0.1807
VideoGames	0.6835	0.6835	0.6835	0.6835	0.6835	0.6835	0.6835	**0.6835**	0.4514	0.173
Average	0.6756	0.6756	0.6756	0.6756	0.6756	0.6756	0.6756	**0.6775**	0.3114	0.1514

Table 5.7: F1 Score of The Naïve Bayesian classifiers under decreasing word usage percentage. The dataset from Amazon.com consists of user reviews from 20 product categories as domains with each domain has about 1,000 reviews.

F1 Score / Model / Datasets	NB-S	NB-T	SU-LML
CableModem	0.4774	0.6633	**0.8694**
Dumbbell	0.6539	0.764	**0.8748**
Flashlight	0.6536	0.6251	**0.8259**
Gloves	0.5973	0.6943	**0.785**
GPS	0.6447	0.7465	**0.9121**
GraphicsCard	0.4797	0.7346	**0.8768**
Headphone	0.5938	0.7356	**0.8858**
HomeTheaterSystem	0.6242	0.8611	**0.9236**
Jewelry	0.6927	0.7088	**0.7599**
Keyboard	0.6905	0.7289	**0.8707**
MagazineSubscriptions	0.6284	0.8056	**0.8932**
MoviesTV	0.4991	0.6785	**0.8381**
Projector	0.6565	0.7525	**0.8575**
RiceCooker	0.6833	0.8027	**0.8475**
Sandal	0.6972	0.6904	**0.8059**
Vacuum	0.7728	0.8	**0.8992**
VideoGames	0.5665	0.7564	**0.9068**
Average	0.6242	0.7381	**0.8607**

Table 5.8: F1 Score for NB-S, NB-T, SU-LML

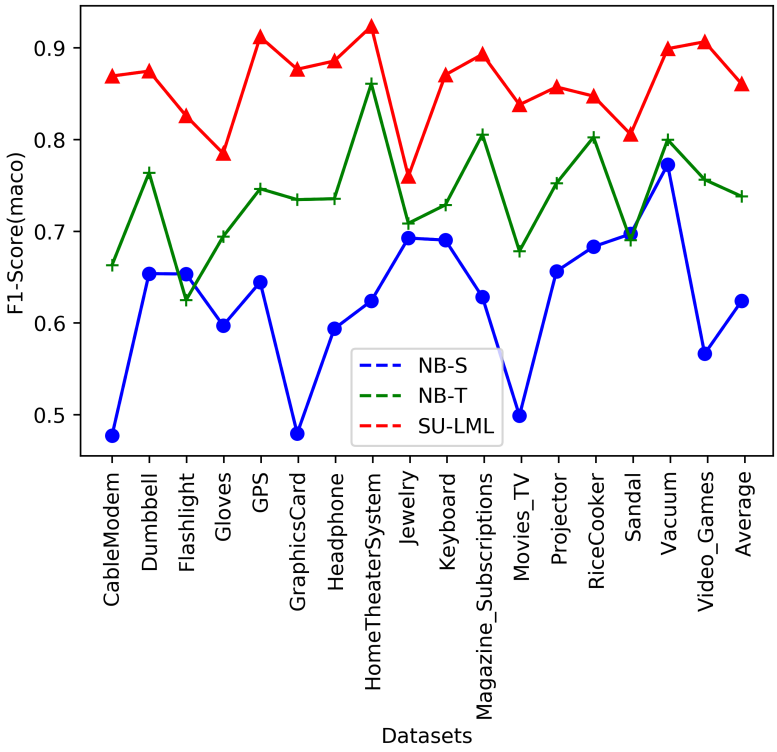


Figure 5.22: F1 Score in Self-Study Stage

Word	Degree for Negative Sentiment
refund	32.99921813917123
garbage	32.994266353922335
junk	32.985405264529575
waste	32.984102163148286
worst	32.97185301016418
rma	32.96846494657285
poorly	32.96194943966641
terrible	32.95569455303623
disappointed	32.949960906958566
trash	32.948918425853535
useless	32.94683346364347
worthless	32.94057857701329
awful	32.92520198071411
defective	32.917904612978894
return	32.913734688558776
exchange	32.908001042481104
respond	32.90487359916601
poor	32.90409173833724
disappointment	32.90278863695596
crap	32.89653375032577

Table 5.9: Top 20 Words with Negative Sentiment

5.9.4 Results

Knowledge generated during learning: If a word is considered to have sentiment polarity, then we increase the polarity score by one. Subsequently, we add an additional score from 0 to 1 based on the $O(w)$ rank. Observe from Table 5.9, that the most top-ranked words are with negative emotion, and most of the predictions are accurate.

5.10 Summary

This chapter introduced an ensemble framework of stream and batch data for an incremental Lifelong Learning framework. The proposed MALA architecture retains the past learned knowledge and transfers it to future learning. The framework can update the dimension-reduced feature set in every streaming sliding window without rebuilding the entire training set from scratch. With the arrival of new datasets over time, the model quickly retrains itself and iteratively becomes more knowledgeable. The RDF based method provides the root cause analysis only on the anomalous cluster identified by the clustering rule, removing unnecessary data clutter and pivoting the actual problem domain much faster. The proposed Decision Tree contains a relatively complex optimization structure and is capable of predicting the profile of future patients. The Decision Tree, however, is erroneous in a few instances for root cause analysis and forecasting. The classification and regression error in the Decision Tree often arises from null entries in patient genotype and phenotype parameters.

In the case study with GDC cancer Patient data, the proposed framework successively applies the streaming clustering technique and RDF Regressor and Classifier to isolate anomalous patient data and provides reasoning through root cause analysis by feature correlations with an aim to contribute to the broader agenda of improving the overall survival rate for patients with prostate cancer. While the stream clustering technique creates groups of patient profiles, RDF further drills down into each group for comparison and reasoning.

In addition to incremental learning methods, this chapter provides a semi-unsupervised *lifelong*

sentiment classification approach. The model can accumulate knowledge from previous learning and turn the past knowledge into self-study. Very few labelled data are required during the training process with this approach. Therefore, the method is suitable for real-world data-intensive applications. The method's performance often exceeds the supervised learning, thus proving that the knowledge reusing with the Lifelong Learning approach is more efficient than a one-shot learning approach. Although the model is validated for two classes of classification, the concept is applicable for multi-classes classification problems. Text classification can apply the proposed approach, and the method is not only limited to sentiment classification.

In the future, it is planned to engage oncologists and clinical pathologists along with data scientists to roll out the proposed framework in practice. Also, *Lifelong Learning through Lambda Architecture* leaves a few interesting open questions to further improve the Lifelong Learning model using the boosting-based ensemble method as investigated in the following chapter.

Chapter 6

Ensemble Learning to Improve Lifelong Machine Learning Results

6.1 Introduction

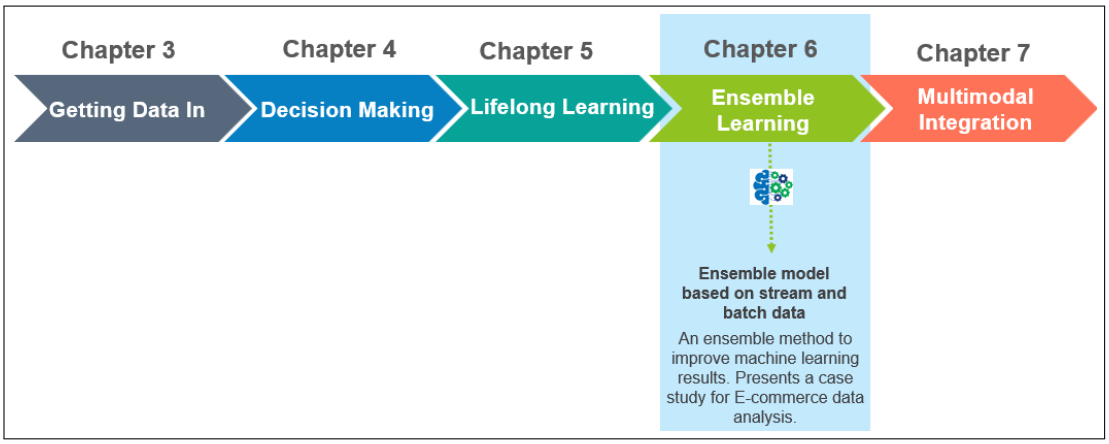


Figure 6.1: An outline of Chapter 6 and end-to-end data flow pipeline through different chapters.

The chapter addresses the key Research Questions 3, 4 about the collaborative ensemble framework for big data and a lifelong learning model through integration between batch and real-time datasets. The chapter extends the lifelong incremental learning model presented in the previous

chapter using the big data hybrid data processing Lambda Architecture, consolidating low-latency real-time frameworks with a high-throughput Hadoop batch framework over a large distributed setup. In this chapter, a *boosting based ensemble method* is presented that improves lifelong learning results by incrementally reducing the *learning error* in each iteration of a streaming window. The chapter addresses the high latency problem of Hadoop MapReduce jobs by simultaneous processing at the speed layer to the service requests requiring a quick turnaround time. At the same time, the batch layer in parallel provides comprehensive coverage of data in terms of accuracy and throughput, blending stream and historical data through the weighted voting method. The cold start situations in streaming services are addressed through an initial offset from historical batch data. Challenges of high-velocity data ingestion are resolved with the distributed message queues.

The efficiency of the batch model is validated by implementing a list of functionalities for e-commerce applications. The critical significance of the proposed model is that the multimodal interaction between batch and real-time components produces deeper analytical insights at low latency. Hence, the proposed system can be highly appealing for applications involving big data, catering to high-velocity streaming ingestion and a massive data pool.

Limitations of the Hadoop framework. Apache Hadoop is the de facto standard for batch-processing systems, used to provide high throughput, comprehensive, and more accurate view of historical data compared to real-time systems. Nevertheless, it suffers from several challenges like high latency, increased operating cost due to larger storage, and bigger cluster requirements for processing one-shot long-running batch jobs. Traditional stream processing frameworks, on the contrary, provide a quick turnaround time using a relatively smaller cluster but lack accuracy and may experience cold start situations [1]. An efficient analytical model should benefit both the *low latency* and *high throughput* requirements through *simultaneous* processing and a mix-processing approach.

This chapter aims to address the current research gaps through an optimized ensemble framework for big data through Multi-agent Lambda Architecture and a boosting based ensemble model.

[1]The cold start problem refers to the situations when the system does not have sufficient information due to sparse data availability in order to make predictions.

The batch and real-time components of Lambda Architecture are two autonomous and collaborative agents. Lambda Architecture (LA) is a new research direction towards the next generation of big data analytics that has started to witness a wide industry adaptation in recent years. However, research efforts towards LA are still in their infancy. *This research claims to be the first general-purpose and comprehensive approach towards hybrid learning of batch and stream data.*

Multi-agent Lambda Architecture. This work is based on the stream-batch mix processing paradigm of the Multi-agent Lambda Architecture (MALA). New data is ingested into both the batch and the stream layers simultaneously. The objective is to serve real-time events with low-latency responses, while a comprehensive analysis of the data is performed through the batch pipeline. See Chapter 5 for a detailed discussion about MALA.

6.2 Summary of Contributions

1. **Ensemble learning to improve machine learning results.** An efficient collaboration between batch and real-time agents can deliver deeper analytical insights at low latency. The model can get more precise over time with incremental learning. The framework produces significant infrastructure cost savings through smaller cluster requirements and shorter training time.

2. **Boosting based ensemble method.** MALA improves lifelong learning results by incrementally reducing the learning error in each iteration of a streaming window using an ensemble method.

3. **Geospatial data analysis.** This work examines the geospatial dataset through Spatio-temporal queries to perform data mining on the volume of records attached to both location and time.

4. **Weighted hybridization strategy.** The chapter introduces a *weighted hybridization* strategy between batch and streaming components. As a principle, the current trend gets more weight over the historical batch data.

6.3 Application Scenario: Case Studies

This chapter implements the proposed ensemble frameworks in *two* different case studies: (i) recommender system and applications on e-commerce domain such as outliers detection, network traffic load, and selling prediction (ii) geospatial-temporal data analysis using New York taxi app data.

6.3.1 Case Study 1: Implicit Feedback Based Recommender System

As proof of the proposed ensemble methods, the case study demonstrates a multimodal collaborative approach that blends real-time stream with historical batch data to produce recommendations with higher accuracy compared to a batch only or stream the only model. The results of experiments indicate that Lifelong Machine Learning results can be further improved through *ensemble* of multiple parameters like context, time, place, etc.

Let us consider the problem of building a personalized recommendation system for an e-commerce portal where the items listed are fairly time-sensitive about the price and other parameters changing over time. The item parameters (e.g. price, category, etc.) are pre-stored in the database, user features are extracted from users' click data (for non-logged-in users), and user profile data (for logged in users) are also saved in the database. The objective is to maximize the number of clicks by showing the most relevant *personalized recommendations* to the users.

This case study introduces a new technique for contextual item-to-item *Collaborative Filtering* based Recommender System, an improved version popularized by e-commerce giant *Amazon* two decades back. The concept is based on items also-viewed under the same browsing session. Users' browsing patterns, locations, timestamps are considered as the *context* and *latent factors* for each user. The proposed algorithm computes recommendations based on users' *implicit* endorsements by *clicks*. The algorithm does not force a user to log in to provide recommendations. It is capable of providing accurate recommendations for *non-logged in users* and with a setting where the system is unaware of users' preferences and profile data (non-logged-in users). In continuation to the previous chapter, this research takes the cues from human *lifelong incremental learning* expe-

rience. It applies these to machine learning on a large volume of the data pool. First, all historical data is gathered from collectable sources in a distributed manner through big data tools. Then, a long-running batch job creates the initial model and saves it to Hadoop Distributed File System (HDFS). An ever-running streaming job loads the model from HDFS and builds on top of it in an *incremental* fashion. At the architectural level, this resembles the big data mix processing *Lambda Architecture(LA)*. The recommendation is computed based on a proposed equation for a weighted sum between near real-time and historical batch data. Real-time and batch processing engines are two autonomous Multi-agent systems in collaboration. A Multi-agent Lambda Architecture (MALA) is proposed for the recommendation engine. A novel *Lifelong Learning Model (LML)* is introduced for recommendation through MALA. The recommender system incrementally updates its model on streaming datasets to improve over time.

6.3.2 Summary of Functionalities

In the case study, a new technique for implicit feedback based recommender system is developed. The main features of the case study are:

- The case study introduces an Implicit Feedback Based Recommender System (IFBRS) using a lifelong machine learning approach. The dataset used is taken from *Movielens*: (https://grouplens.org/datasets/movielens/100k/).
- The work redefines the item-to-item relationship by assigning more relevance to the current trend than the historical data. A hybrid method combines the results from Collaborative Filtering with content-based filtering on item similarity.
- As an implicit feedback (clicks) based recommender system, it interprets item-to-item collaboration in one browsing session by each user, which is in contrast to the traditional user to user similarity approach.
- To make recommendations relevant, IFBRS extracts the user's browsing location as a latent factor. Context-aware recommendations tend to be more accurate.

- This case study redefines the item-to-item relationship by putting more relevance on the current trend than the historical data.
- The case study introduces a novel architecture of an end-to-end recommender system with a host of online and offline big data ecosystem tools and their correlation as a multimodal interactional model. The proposed architecture can be extended to other big data domains taking advantage of both stream and batch processing capabilities.

6.3.3 Functionalities at the Stream Layer

Along with the recommendation service, which is served at the batch layer, the following functionalities *simultaneously* takes place at the streaming layer.

- Detecting the *outliers* in the user buying patterns or network usage.
- Forecasting network traffic load for the e-commerce portal.
- Identifying users buying patterns through the information trail from viewing to buying an item and compute the moving average of the items checkout volume.

6.3.4 Case Study 2: New York Taxi App Data Analytics

A second case study examines the New York taxi trip dataset with analytical and predictive insights about people's mobility patterns and landscapes. The case study uses the MALA to simultaneously offering real-time and batch functionalities.

6.4 Ensemble Learning in MALA

This section discusses ensemble learning in MALA through batch and stream processing. In the traditional big data systems, feature vectors initialize themselves with historical batch data and persist data into Hadoop Distributed File System (HDFS) during the training phase. A long-running batch schedule trains the vector, and responses are again stored in HDFS. One of the primary

constraints with this setting is that the full dataset may not be available at the initiation of a training schedule. The other key challenge is, the vectors can be updated with new events coming over time. Therefore, an iterative method is proposed that transforms the model training process into a *lifelong learning machine* [248] [274]. The Streaming model initializes itself with *saved learning* with the batch mode by loading the trained model from the Hadoop Distributed File System (HDFS) into a distributed memory. The MALA updates its streaming model incrementally on each new wave of incoming data. The framework further allows the merging of static historical data pool with the most updated streaming model. The method removes any cold-start situations at the training initiation. Figure 6.2 shows a screenshot of the learning process running in the command line.

Figure 6.2: Screenshot showcasing the incremental streaming learning. MALA initializes with historical batch data and update the streaming model incrementally on each new wave of incoming data. Model runs indefinitely at a 10 seconds window interval. Training folder is updated incrementally with new data (Figure a). Model is retrained simultaneously with a small amount of new data as they appear (Figure b). Figure b shows the sliding window interval for the model retraining every 10 seconds. The training starts with multiple stages as soon as new dataset is copied to the training folder.

Let us consider a scenario with the entities *users* and *items*, in an e-commerce portal. The entities are represented as high-dimensional feature vectors. The Correlation between the user and item (r_{ui}) in the shopping portal can be defined as:

$$f(r_{ui}) = v_u v_i \tag{6.1}$$

v_u and v_i are the feature vector for the user and item respectively. Modelling the interaction between high-dimensional feature vectors (user and item) has a significant limitation of responsiveness, especially with big data. Also, the vectors are updated with new raw events arriving over time. Therefore, the constant need arises to revise the feature vectors leading to develop an iterative

approach. With the iterative approach, the vectors initialize with historical batch data and persist them into Hadoop Distributed File System (HDFS). A stream processor loads the model from HDFS and incrementally builds on top of it. So, the Equation 6.1 is updated to accommodate the incremental (delta) learning as follows:

$$f(r_{ui}) = v_u v_i + \delta_u \delta_i \tag{6.2}$$

The model learns v_u, v_i offline and δ_u, δ_i at an iterative fashion with small amount of streaming data. δ_u and δ_i are the most current updates or fresh new rows of data. The dimensionality of δ_u (or, δ_i) is the same as v_u (or, v_i). However, the data size is much smaller, only the volume of most recent data collected at a streaming window length of few minutes and therefore not very large. The method allows the model to keep updated at low latency on a small amount of incremental data.

A *dimension reduction* approach can be taken for vectors [35], producing faster training time. So, δ_i or δ_j need not be of the same dimension with the user or item vector. Instead, only the modified columns can update the training model through the online process. The approach reduces the dimension of streaming learning data and the online learning time since the online model only needs to learn the *correction* over the batch offset. Nevertheless, in the current dataset, the search for the rows and columns that requires an update and subsequently merge the incremental result with batch is computationally expensive on a large volume of the data pool. Hence, the approach to *retrain* the model on a small window interval of data pool through distributed in-memory processing of Apache Spark provides faster response time. The lifelong learning method is described in Algorithm 5.

6.5 Incremental Boosting-based Ensemble Proposal for MALA

This section presents an ensemble-based boosting method that can be appealing for the MALA to improve lifelong learning results. In a non-stationary environment, a large pool of historical

Algorithm 5 Lifelong Hybrid Learning Algorithm

Input: historical batch datastore d_b, dataset as collection of streaming records since last window $d_s = \sum_{i=1}^{n} d_i$, Kafka topic k_t, DB server address d_a,

Output: : Updated knowledge base kb

1: $C_b \leftarrow$ BatchComputationRule(d_b)
2: call SubscribeKafka(k_t, d_a);
3: **for** windowed dataset d_i **do**
4: Spark consumes Kafka queue
5: $C_s \leftarrow$ StreamComputationRule(d_i)
6: $C_{bs} \leftarrow$ Update(C_b,C_s) //update batch with stream
7: kb $\leftarrow$ C_b, C_s, C_{bs}
8: **return** kb //kb is persisted into distributed memory and HDFS in data blocks
9: **end for**

datasets is accumulated, and new data being appended all the time at a very high rate. MALA aggregates the weak *base learners* by adding successive mini-batches. Every window of a dataset eventually becomes *stronger* than its previous iteration. The task can be achieved by computing on each record at an *ideal* streaming setting using big data stream processing frameworks such as *Flink* [56]. The case study with the classification of e-commerce data (Section 6.11), *Apache Spark DStream* mini-batches are used at a near-real-time window size of a few hours. The model is required to update with new data as fast as possible, not necessarily exactly in real-time. The experimental results in Section 6.8 highlight better performance in terms of processing speed, throughput, and accuracy with near-real-time Dstreams mini-batches than real-time *Structured Streaming*.

6.5.1 Streaming Ensemble Proposal

The streaming ensemble method incrementally *boosts* the accuracy on each iterating mini-batch, enabling the model to accumulate knowledge much faster. The *base learners* adapt more quickly in smaller intervals of a sliding window, improving the accuracy rate by countering the *concept drift*. In each iteration, a weak learner is boosted as a weighted sum of previous learners as defined

by Equation 6.3.

$$S_L(.) = \sum_{n=1}^{L} c_l \times w_l(.) \tag{6.3}$$

Where $S_L(.)$ is the strong learner, c_l are the coefficient for each of the weak learner $w_l(.)$. Instead of computing on the entire dataset as a single batch, an incremental stream learning method is more efficient in a non-stationary environment. The iterative boosting process is identical to a *gradient descent* problem [51] [53]. At each streaming window, we can fit a weak base learner from the previous iteration to the gradient negative to the current iteration (details below). Gradient descent method for ensemble over stream data is given by:

$$s_l(.) = s_{l-1}(.) - c_l \times \delta_{sl-1} E(S_{l-1}).(.) \tag{6.4}$$

Where c_l is the coefficient, $E(.)$ is a fitting error at an iteration, $E(S_{l-1}).(.)$ is the negative of the gradient for the fitting error (residuals) as compared to the previous iteration.

Steps for the proposed ensemble model on streaming dataset are as presented below:

Step 1 Start with streaming window and initial residual error e_i, measured as distance from the cluster center.

Step 2: Fit a Decision Tree or Linear Regression function on x_i input data, y_i is a target output, p_i a predicted output, e_i is the fitting error.

Step 3: Compute the fitting error as predicted *minus* target value. So, $e_i = y_i - p_i$

Step 4: On a new streaming window, take e_i as target variable, fit a new model and predict a value p_{i+1}, and a new residual e_{i+1}.

Step 5: Add the residual error to the previous predictions. So, $P_{i+1} = P_i + e_{i+1}$

Step 6: Check the accuracy as $y_{i+1} - p_{i+1}$, validating with the actual value to detect overfitting.

Step 7: On a new streaming window, fit another model on the residual e_{i+1} and iterate the steps 2 to 6 if residual changes from the previous window and model is not overfitting as per *step 6*.

Figure 6.3 summarizes the steps.

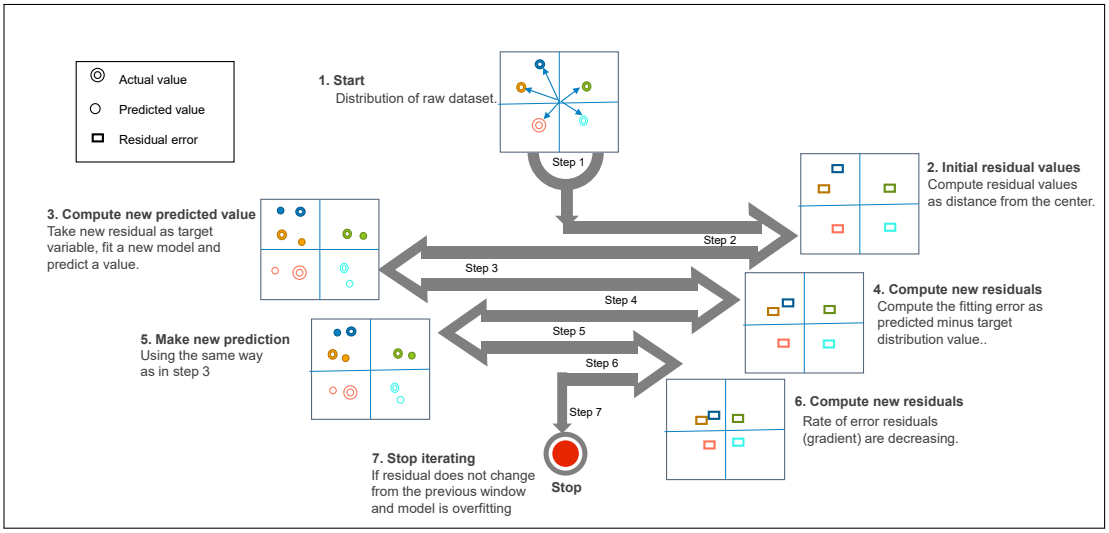


Figure 6.3: End-to-end flow of the ensemble boosting processing on a streaming environment.

The Figure 6.4 shows the classification results based on *one-time* learners like Decision Tree (DT), Random Decision Forest (RDF), and *boosting based proposed method* using DT and RDF as the respective base learners. Base learners classify data depending on a feature threshold that divides the space into *three decision surfaces*. The Figure illustrates that the *boosting* based techniques with appropriate decision surfaces can improve accuracy. The most accurate results are obtained with the RDF based booting method.

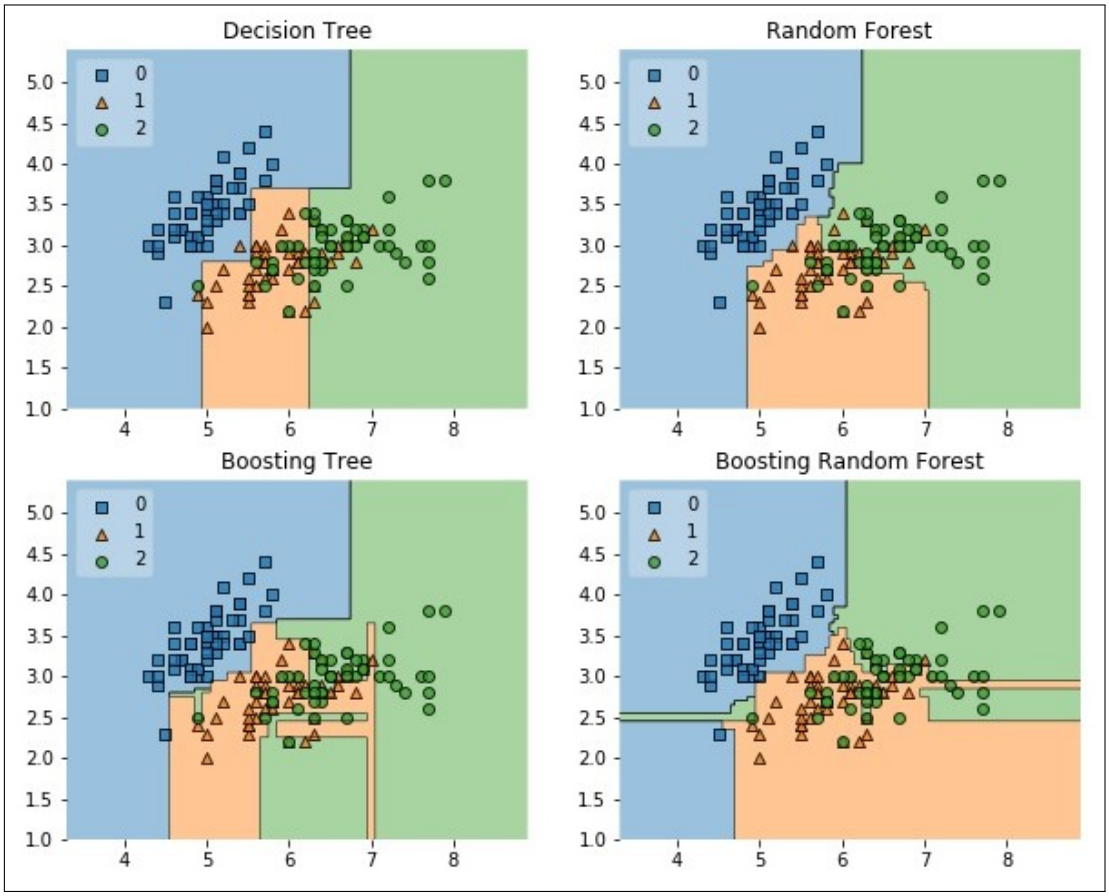


Figure 6.4: Classification results of proposed methods on *iris dataset* consisting of samples from Iris flowers of three related species under five attributes - sepal length, sepal width, petal length, petal width and species [243]. As shown in the Figure, boosting based methods are more accurate compared to one time learning methods such as Decision Tree or the Random Decision Forest.

6.6 Case Study-1: Building a Recommender System through the Ensemble Model

The MALA enables simultaneous execution of batch and real-time view that collaborates with each other to present a more comprehensive and accurate view. In that context, this section presents a novel architecture of an e-commerce Recommender System under batch settings using the MALA

as its primary processing component. The framework computes *Recently viewed items* by each user under *streaming settings* that influences the final recommendation order computed under *batch settings*, making it an ensemble model between the batch and the stream. A *weighted hybridization* strategy of multiple ensemble parameters decides the final recommendation score.

6.6.1 Recommender System Introduction

Recommender systems are a useful alternative to search algorithms since they help discover items in e-commerce sites users might not have found themselves. Recommender systems adopt three types of approaches: (a) Content-based, (b) Collaborative (c) Hybrid. Content-based filtering methods are based on a description of the item and profile of the user's preferences [288]. The system creates a content-based profile of users based on a weighted vector of item features. These are often combined with the user's feedback to assign a higher or lower weight to order the *top n recommendation*. For example, when a user views a mobile phone in the shopping portal, a few other similar mobile phones are recommended for him. On the other hand, Collaborative Filtering methods [48,100,131,144] are based on collecting and analyzing a large amount of information on all users' behaviours, activities, or preferences and predicting what users will like based on their similarity to other similar users or items. The underlying philosophy is on the assumption that people who agreed in the past will agree in the future. The hybrid approach makes content-based and collaborative-based predictions separately and blends them with weightage.

Recommender systems get input from different sources to make recommendations. The most common way of collecting input is through the user's feedback. Shopping portals, for example, collect ratings provided by the users. However, explicit feedback may not always be available. Thus, several attempts were taken to build recommendations through users' implicit feedback [112].

The recommender system in e-commerce provides a prominent way to enhance the overall shopping experience by delivering personalized or contextual advice through mining and discovering the interests and analyzing patterns of customers. The method has made e-commerce more personal. No longer are products marketed to a mass audience, but individuals based on their

unique needs. It also has a significant role in generating revenue for the website by the fact that users tend to purchase if a recommended product is relevant to his need.

6.6.2 Computing Recommendations through Collaborative Filtering

The primary motivation behind item-based recommendation algorithms depends on the fact that the two items viewed by the same user in the same session are likely to be related to each other. Collective co-occurrence count sorted in descending order for each item reveals the trend of which items users are likely to search together most. The associative or *collaborative filtering* rule is the central theme of the implicit feedback based recommender. The idea of a collective count of items co-viewed under the same browsing session across all user bases eliminates any reliance on user-provided ratings. User-provided ratings are sparse, but users' *click to view* data are always plentiful, and therefore the model is useful in sparsity and cold start situations.

6.6.3 Model Training

Let f_i be the feature vector of item i and f_u is the feature vector of user u. v_u denote the latent factor vector of user u and v_i is the latent factor vector of item i. When user u interacts with item i, CTR is predicted as $1/(1+exp(-R_{ij}))$ by the Logistic model. Where the response score R_{ij} is given by

$$R_{ij} = f_i K f_u + v_i v_u \tag{6.5}$$

K is the regression coefficient matrix representing user-item interactions, extracted and learned from the input dataset. In implicit feedback-based systems, both latent factors v_i and v_u are frequently updated since parameters are derived from clickstream data.

Offline Learning

Offline learner updates the model parameters and a latent factors (K, f_i, f_u, v_i, v_u) from users clickstream data. The processing of a large data volume involves a parallel processing framework

like Apache Spark. The offline process requires long-running jobs to complete, which executes every few hours interval. It ingests the trained models and parameters to the storage system using the Apache Flume [128] [208] from where the online learner incrementally updates the model.

Online Learning

Online learners update the model parameters in real-time at a sliding window protocol to keep the model parameters in sync with the most recent changes. For known parameters, K, f_i, f_u, v_i, v_u; the online learning is about approximating a large number of independent regression problems for each individual item in i. The online regression process estimates v_i and v_u, while keeping $f_i K f_u$ as an offset. The incremental online learning process is described in detail in section 6.4.

6.6.4 Synchronizing Offline and Online Learning

The offline process is scheduled to initiate at an interval, 6 hours in the current setup. When the offline learner finishes scheduled jobs, it stores the learned parameters K, f_i, f_u, v_i, v_u into a NoSQL storage. While the offline version provides a more comprehensive model training through the MapReduce framework, the online process keeps the model up to date with a near-real-time retraining process. The offline model is updated with newly learned model parameters in each iteration of a batch run, and the current iteration serves the end-users using the online model. The online process subsequently discards the old offline model and updates on top of the most recent version of the offline model.

6.6.5 Big Data Solution Enabling Rapid Aggregation to Build Item Co-occurrence Matrix

The recommender system stores the stream data into the Cassandra DB. A co-occurrence job using Pig scripts or Java MapReduce is initiated at a periodic interval by a *Apache Ozzie* scheduler, generating the *items co-occurrence matrix*. Items for individuals users are collated and subsequently merged with all other users' click to view data at the following sequence:

User views items A, B, C:

A-B (and B-A)

A-C (and C-A)

B-C (and C-B)

We can initialize the co-occurrence count to *one*. See Table 6.1

Table 6.1: Co-occurrence Table

item1	**item2**	**count**
A	B	1
A	C	1
B	C	1
B	A	1
C	A	1
C	B	1

Assume, users continue to view items *B, C, D* under the same browsing session. We can create or increment the association by count *one* and create the co-occurrence matrix for a single user:

B-C (and C-B)

B-D (and D-B)

C-D (and D-C)

The count field is updated with each iteration. See Figure 6.5.

The process is repeated across the all user base and is merged to create a single co-viewed matrix. Results of the co-viewed matrix may look like this: *iPhone, Galaxy phones viewed together 100 times; iPhone, LG phones viewed together 80 times at the same session by all users*. Note, items viewed under the same browsing session helps to compute the related items searched by past users and, therefore, can be recommended to future users.

item1	item2	count
A	B	1
A	C	1
B	C	2
B	A	1
C	A	1
C	B	2
B	D	1
D	B	1
C	D	1
D	C	1

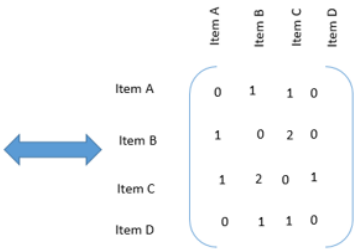


Figure 6.5: The final Co-occurrence matrix based on items viewed together under the same browsing session across all users base.

6.6.6 Computing Recommended Items

A weighted hybridization strategy combines two or more factors by computing weighted sums of their individual recommendation scores.

Following are the different latent factors:

- Co-occurrence count
- User's location
- User preferences
- Timestamp of the click data in the recently-viewed table.

Co-occurrence count is obtained in the previous step. The following section describes latent factors: location, user preferences, and timestamp. The overall recommendation score is computed for a user by Equation 6.6:

$$R = \prod_{k=1}^{n} \omega_K \tag{6.6}$$

ω_K is the weight of each latent factor. Weight for co-occurrence count is normalized through the following equation:

$$\overline{\omega}_K = \frac{\omega_k}{max_k(\omega_k)} \tag{6.7}$$

where ω_k is the *view count* of the kth product and $max_k(\omega_k)$ is the maximum *view count* across all categories the user viewed. In the experimental setting, cumulative *view count* is considered for a period of *two weeks*.

6.6.7 User's Location

An *active user* is the one for whom the recommendation is computed based on all passive users' collective historical records. First, *active user's* physical location is retrieved from click data by parsing *click to view* stream URL. Then, to compute the recommendation score, a higher weight is applied for the same location where an item is viewed, and few other items are *also-viewed* by the *active user* under the same browsing session.

Suppose, if an active user is based at location l_1 and recommendations are shown for *viewed item* i_1, then we can compute *also-viewed* items for i_1 as i_2, i_3. If we consider i_1 and i_2 are viewed in the same location, we can assign more weight to the score of i_2. If i_1 and i_2 are viewed together n times and both are from the same location then the recommendation score for i_2 = n$\times l_1$, where l_1 is the weight applied for a similar location. If i_1 and i_3 are from two different locations and the occurrence count between them is m then recommendation score is calculated for $i_3 = m \times l_2$, where l_2 is the weight applied for different location.

6.6.8 Users' Preferences

The implicit feedback-based recommender is not reliant on the user's explicit preference settings that the user might have set while creating a profile with the portal. Rather, in an implicit model, preference is expressed as the user's *confidence* on a particular category of the product measured by repeated views by the same user. Suppose, a user views *smart watches* multiple times which is under subcategory of *wearable device* under category *electronics*. Then, recommending the user a few smartphones is a quite simple and effective strategy. We can apply greater weight to the

same sub-category of recommended products in which the user shows more *confidence* by multiple views. View count is normalized by Equation 6.7.

6.6.9 Weighted Hybridization Strategy through Time-variant Data

In an e-commerce scenario *age of the data,* defined by the time order user viewed the items, can play an important parameter for relevancy of recommended items [273]. *Data age* is therefore an influencing factor ranking the *top n recommendations*. IFBRS prioritizes recent data over old historical data. To achieve this, first, we compute the list of recently-viewed items from Cassandra and order them reverse chronologically so that the most recent item is placed at the top of the list. Thereafter, a big data solution enables rapid aggregation to create a co-occurrence matrix for each recently-viewed item, as discussed in the preceding section. Since recently-viewed items are reverse chronologically ordered, when weight is applied in descending order, most recent items end up receiving higher weights. Real-time and historical batch processing components are two collaborative autonomous Multi-agent Systems. See Algorithm 6.

6.6.10 Illustrative Example

Assume, a user u_1 is in a location *loc-1*, opens an e-commerce portal and views 3 items: *A, B, C* at time t_1, t_2,t_3. Where $t_1 < t_2 < t_3$. Recommendations are built for the active user u_1.

First, we consider historical data pre-loaded into the database capturing cumulative click data from the entire user base. We can use the past dataset to create the *also-viewed* table as shown in Figure 6.5. The table contains items *also-viewed* across all user bases. A periodic batch job aggregates this data by adding the also-viewed count. For simplicity, we consider user preference count is *one* for all products; that is, the user has viewed items *A, B, and C* only once. See Table 6.2.

The recommendation score is computed based on Algorithm 6. A weighted hybridization strategy computes the final score by applying different weightage schemes on location and time. If the item *viewed* and *items also-viewed* are from the same location, higher weight is applied to *also-*

Algorithm 6 Implicit Feedback Based Recommender System

Input: User generated click stream event, time-decay-factor

Output: : Ordered list of recommendations with descending score

item_decay_factor=y, item_timestamp_weight=x,
for each user u who *viewed* item I_i **do**
  Compute a reverse chronological list L_i of all the items recently-viewed I_i (i=1 to n) in
  for each L_i **do**
    Compute *also-viewed* item by user u as related item RI_i under the same browsing session
    for each item pair I_i ,RI_i **do**
      $M_{ij} \leftarrow$ countCooccurrences(I_i, RI_i) //M_{ij} is the co-occurrence matrix between pair I_i, RI_i
      $R \leftarrow top\_n\_Items(M_{ij})$
      $R \leftarrow R \times \omega_l$ //applying location factor
      $R \leftarrow R \times \omega_p$ //applying user's preferences factor
      $R \leftarrow R \times \omega_{\bar{c}}$ //applying normalized co occurrence count factor
      $R \leftarrow R \times \omega_t$ //applying timestamp factor
      $\omega_t =$
      ω_t - $time\_decay\_factor$
    end for
  end for
end for
$R \leftarrow sort(R)$ //Final recommendation order of I_i ,RI_i
return R //Ordered list of recommendations

Table 6.2: Counting top *n* recommendations: Step 1

Timestamp	**Item viewed**	**User preference count**	**Item also viewed**	**Similarity strength**	**Also viewed count**
T1	A	1	D	0.98	10
T1	A	1	E	0.95	9
T1	B	1	F	0.88	11
T1	B	1	G	0.97	12
T2	B	1	H	0.77	15
T3	C	1	I	0.95	8
T3	C	1	J	0.94	6

viewed items. Similarly, for the *timestamp* parameter, weights are adjusted in reverse chronological order. Thus, items with the latest timestamps are assigned to the highest weight. See Table 6.3.

Table 6.3: Counting top *n* recommendations: Step 2

Item also viewed	**Also viewed count (norm) (c)**	**Item viewed location (l)**	**Also viewed location (l)**	**Recommendation score** ($t \times l \times c \times p$)
D	1	Loc1	Loc1	$1 \times 1 \times 1 \times 1 = 1$
E	0.9	Loc1	Loc2	$1 \times 0.8 \times 0.9 \times 1 = 0.72$
F	0.73	Loc3	Loc3	$0.9 \times 1 \times 0.73 \times 1 = 0.65$
G	0.8	Loc3	Loc4	$0.9 \times 0.8 \times 0.8 \times 1 = 0.57$
H	1	Loc3	Loc5	$0.9 \times 0.8 \times 1 \times 1 = 0.72$
I	0.61	Loc6	Loc6	$0.8 \times 1 \times 0.61 \times 1 = 0.48$
J	0.46	Loc6	Loc7	$0.8 \times 0.8 \times 0.46 \times 1 = 0.29$

timestamp *t*, location *l*, count *c*, preferences *p*

Based on the recommendation score, final order for the *top n recommendations* are:

D>E>H>F>G>I>J

6.6.11 A Hybridization Strategy: Computing Recommendation through Item Similarity

The IFBRS overcomes user/item cold-start situations, sparsity, or scalability problems caused by *sparse rating data* available. However, where no historical *click view data* is available for a product (maybe a newly launched product), the user can not see any recommendation since there are no similar items viewed together across all user bases. In such scenarios, the IFBRS can be extended to a *hybridization strategy* to recommend similar items based on the item features. If the IFBRS does not return any item, the item similarity approach is pursued. In such scenarios, the IFBRS returns recommended items, then the items *not* similar to the viewed items are eliminated from final recommendations. Therefore, if a user views a smartphone, a few other *smartphones* appear to him as recommended items and not *shoes* or *watches* which are not similar to a smartphone by category. Two methods can be adopted for item similarity: the first option is just to recommend the same category of products. That is, if a user views a *smartphone*, the recommendations are a few other *smartphones* ordered by popularity.

The second approach creates a user feature vector f_{it} and item feature vector f_{jt} at time t. The method needs to find the similarity between these vectors. There are several ways to measure the similarity between vectors. Starting with a simple setting where f_{it} and f_{jt} are data points in the same vector space; that is, both users and items are represented using the same set of features. We can measure the cosine similarity between two vectors. Nevertheless, the IFBRS does not consist of many users features since the system is designed for predicting recommendations with *non-logged in* users. In reality, users expect to see recommendations in the same browser and device they use without the need to log in. Users tend to log in only during the checkout process. Thus, recommender systems can *not* have access to user features like gender, age, address, etc. Therefore, we can continue on the *method one* which only builds upon the item features. The approach is computationally attractive, too, because of its simplicity and accuracy.

6.6.12 Computing Item Similarity

We compute the cosine similarity between items by comparing identical features based on the item category. During the listing of items in the online portal, each item is placed into a category manually. Hence, item *noodle* goes to the food category and the fast-food sub-category, while the *smartwatches* are placed into a wearable device under the electronics category. Item similarity attempts to find how suitable it is to recommend a user *noodle* if he views *smartwatch*.

The similarity between items is calculated based on how similar their categories are. A novel, computationally faster approach is introduced to achieve this. A numerical category value is assigned against each item. All electronics items are assigned into category values from 1 to 200; food items are assigned into category values of 201 to 400, and so on. Computing cosine similarity on numerical categories reveals how related the two items are. For instance, consider computing the similarity between a television and a camera. A television (product A) may fall under the following category: Appliances (category id 1) - LCD (category id 4) - 3D LCD (category id 6). And a camera (product B) is under the following category: Electronics (category id 1) - camera (category id 35) - DSLR camera (category id 95). We can calculate the item similarity strength of product A and B at a scale of 0 to 1 by the following equation:

$$S_{\mathrm{A,B}} = cos(\theta) = \frac{(A \times B)}{(|A||B|)} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sum_{i=1}^{n} \sqrt{A_i^2} \times \sqrt{B_i^2}} \tag{6.8}$$

$S_{\mathrm{A,\,B}}$ is the similarity between product A and B, $cos(\theta)$ is the angle between edges A and B.

Using Equation 6.8 we can calculate similarity strength between a television and a camera represented as:

A = (1, 4, 6)

B = (1, 35, 95)

Numeric values represent feature vectors of items A and B.

Cosine Similarity $S_{\mathrm{A,B}} = \frac{1\times1+4\times35+6\times95}{\sqrt{1^2+4^2+6^2}\times\sqrt{1^2+35^2+95^2}} = 0.964$.

6.7 Lifelong Learning Model for Recommender System

The recommender system uses the lifelong ensemble learning model between stream and batch as described in section 6.4. Lifelong learning has brought several benefits to recommendation accuracy. The recommendation method described in the preceding section can be extended using lifelong learning to update recommendations to the newly arriving dataset at faster intervals. The knowledge Miner component of MALA enables an incremental learning environment. Algorithm 6 presents the novel lifelong learning method for a recommendation engine using incremental updates.

Algorithm 7 Recommendation Using Lifelong Incremental Learning Algorithm

Input: historical datastore for batch d_b, collection of streaming dataset for the last *sliding window* $d_s = \sum_{i=1}^{n} d_i$, Kafka topic k_t, DB server IP d_a, Knowledge Base k_b

Output: : top n recommended product list

1: $C_b \leftarrow$ CollaborativeFiltering(d_b)
2: $S_b \leftarrow$ ItemSimilarity(d_b)
3: subscribeKafka(k_t, d_a);
4: **for** windowed dataset d_i **do**
5: Storm consumes Kafka queue
6: $C_s \leftarrow$ CollaborativeFilteringRule(d_i)
7: $R_c \leftarrow$ Update(C_b, C_s) //for the first loop
8: $R_c \leftarrow$ Update(R_c, C_s) //collaborative filtering results are updated with stream
9: $S_s \leftarrow$ SimilarityRule(d_i)
10: $R_s \leftarrow$ Update(S_b, S_s) //for the first loop
11: $R_s \leftarrow$ Update(R_s, S_s) //item similarity results are updated with stream
12: R $\leftarrow R_c \cap R_s$
13: $k_b \leftarrow R_c, R_s, R$
14: **return** k_b
15: **end for**

Step 1 (Batch processing rule, lines 1-2): Collaborative Filtering and items similarly based

rules compute the initial recommendations.

Step 2 (Stream processing rule, lines 7-9): Two forms of recommendations are computed at the streaming setting. Their respective results are merged with the batch counterpart. The method makes the model agile to update at shorter intervals.

Step 3 (Computing the final recommendations, lines 10-13): The elements common between Collaborative Filtering and similarity rule are preserved, and the rest are discarded. The final training model is added back to the datastore for future use, and the Knowledge Base is subsequently updated.

6.8 Experiments

6.8.1 Setup

Users *click-to-view* data is high in volume and velocity compared to data generated only from purchases. A robust big data infrastructure is proposed to support the enormous storage and processing requirement. Experiments are carried out in *Amazon Cloud Services (AWS)* for Cassandra, Kafka, and Hadoop installation. The AWS instance detail is shown in Table 6.10. The Apace Storm is installed in the *Microsoft Azure cloud*. In a context-aware recommendation system, the big data event capture framework ingests every end-user click from a customer touch-point. The customer's location and event time are extracted from clickstream and appended into each session object. A session object is created each time the user newly opens the e-commerce website. For each click event, a *session ID* and a *context ID* is created. The context is created from the session object at the JavaScript layer. *Context ID* is associated with each user's click event saved in the datastore for computing recommendations.

Table 6.4: AWS Instance Types

Instance Type	Instance Count	vCPUs	Memory	Instance Storage	EBS Optimized Bandwidth
m1.large	3	2	7.5 GB	500 GB	Moderate

Users' click data, i.e. clickstream is one of the primary sources for deriving implicit feedback. Clickstream contains the user's location information which is derived from the client IP address. Country, region, state, and city are part of location data. Each user-click on the website items generates an event and clickstream, which is captured by Apache Kafka message queue and data is moved from the source system to the analytics processing system on a near real-time basis. Apache Storm processes data in real-time and stores them into a Cassandra data store. Context-aware real-time stream is stored in Cassandra and processed through a batch job running periodically at an interval of 6 hours. The batch job creates an item recommendation table stored in the Cassandra database. The recommendation table is accessed from a UI using RESTful API at the back-end, developed in Java Spring framework. The end-to-end big data ingestion framework is shown in Figure 6.7.

Kafka is a popular choice for real-time data retrieval. Kafka is capable of ingesting a large volume high-velocity of data that requires fast, fault-tolerant, distributed pipelines. As a massively distributed client-server-oriented publisher-subscriber messaging system, Kafka replaced the traditional message queue systems like Rabbit MQ, IBM MQ because of higher throughput, reliability, and replication capabilities in big data scenarios. Kafka is the central hub for real-time processing along with Storm stream processing APIs [19]. Refer to Figure 6.6 for the Kafka cluster used in the data ingestion layer.

6.8.2 MovieLens Dataset

The IFBRS is examined on a real-world dataset from movielens.org [20], and performance is compared with Spark MLlib ALS API [271]. The *Movielns data* contains up to 27,000 movies by 138,000 users. Since the IFBRS does not predict the ratings, instead it ranks the recommended products on the overall score. Subsequently, ranks are validated against the ratings predicted by Apache Spark ALS API.

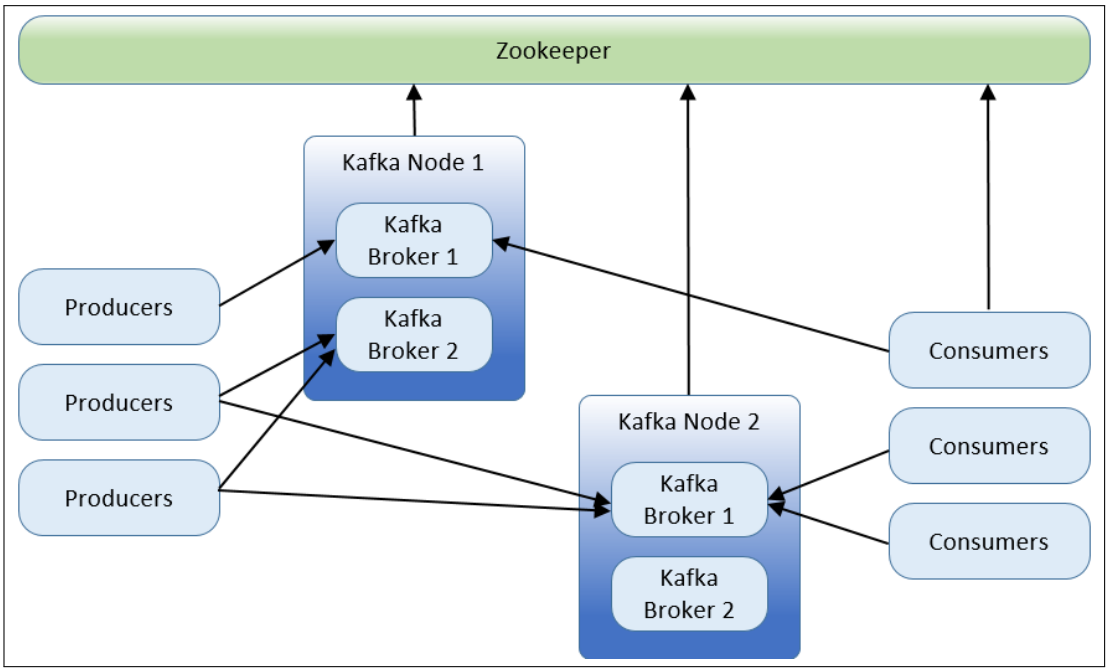


Figure 6.6: Multi-node, multi-broker Kafka cluster. A *Broker* is the actual Kafka process. Producer ingests data into multiple broker components for load distribution and parallel processing. The distributed architecture provides a robust failover and faster processing through load balancing.

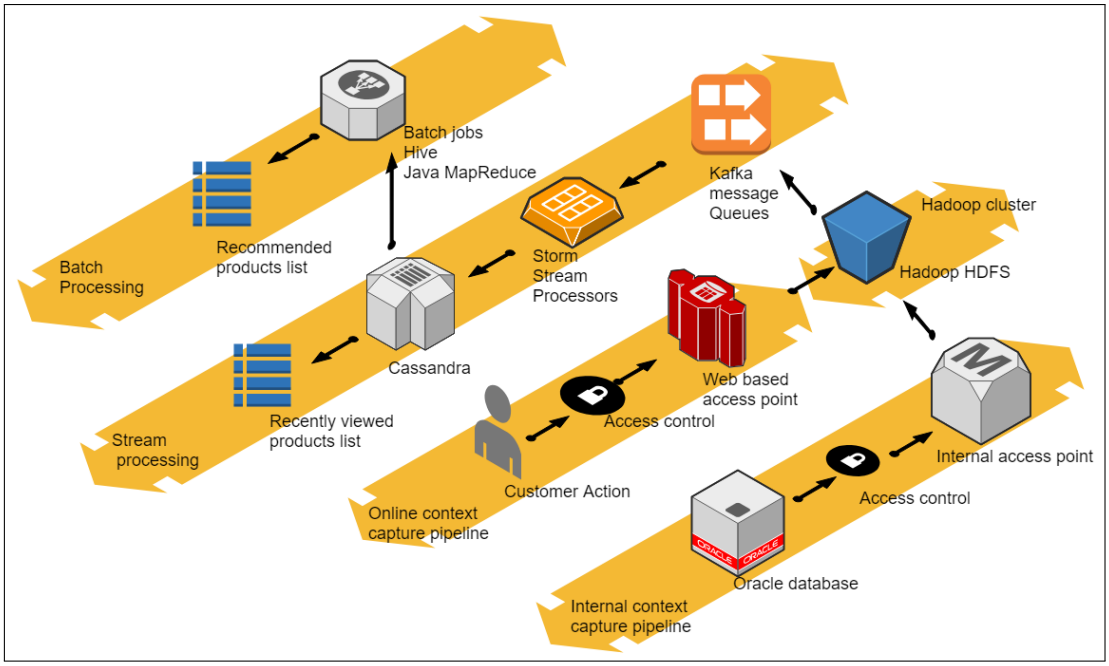


Figure 6.7: End-to-end flow: Ingestion to the recommendation. Each user's click on an item generates an event and clickstream, which is captured by Apache Kafka message queue and data is moved from the source system to the analytics processing system on a near real-time basis. Apache Storm processes data in real-time and persists them into a Cassandra data store. An online context-capture pipeline stores users click data, while an internal context-capture pipeline stores item features as historical data as recorded manually for each item. An efficient blend of historical batch data with a real-time stream creates a lifelong learning environment for a recommender system.

6.8.3 Setting a Storm Cluster

For the Storm setup, the *Microsoft Azure HDInsight* Storm cluster is used for setting up a *nine* nodes Linux setup on Ubuntu OS. Unlike Farahabady [144], a heterogeneous Storm cluster was not considered for Nimbus or Supervisor to make the most out of the default scheduler and load balancer. See Figure 6.8

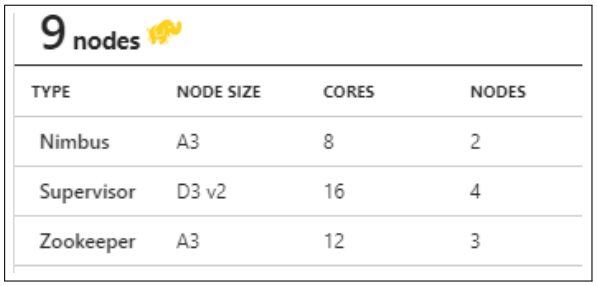


Figure 6.8: Microsoft Azure HDInsight Storm cluster.

The recommender application is developed as Java Spring REST Web service and deployed into Tomcat through Datastax driver [100], which is a popular Cassandra Java client driver.

The Movielens DB has three tables with schema shown in table 6.5

Table 6.5: Movielens Database Schema

Table	Column Names
rating	UserID, MovieID, Rating, Timestamp
users	UserID, Gender, Age, Occupation, Zip-code
movies	MovieID, Title, Genres

6.8.4 Data Preparation

Table schema 6.5 is used to create a user-movie table containing the following column: UserID, MovieID, and Zip-code. Hadoop Pig scripts transform data consisting of *movies also-viewed* across all the user bases through the MapReduce. However, in the current model with the *Movielens Database*, we cannot determine if each user's *also-viewed* products are generated in a single session

or multiple sessions since the algorithm is based on the assumption that items are considered *also-viewed* only if those items are viewed in the same browsing session. Therefore, it is a known limitation with the current data source that can be overcome with the availability of production clickstream data. The user's *location* is the *state* from where clickstream data is generated. The *state* as location is computed from the zip code. Also-viewed products are computed based on the scenarios where products are viewed, and products also-viewed are from the *same or different location* (State). Based on that, Algorithm 6 computes the *top n recommended products*.

6.8.5 Evaluating IFBRS

Evaluation of implicit-feedback based recommender systems can be tricky. Numeric scores are available in explicit rating-based recommender systems as a benchmark, and precise accuracy can be measured using the root mean squared error. However, evaluating the success of an implicit model can not use traditional evaluation methods in the absence of ratings. The implicit model rather can use *users' reaction* to the recommendation by clicking or viewing the recommendations. The implicit model can as well compare the ranking orders of the recommendations with actual ratings provided by the user in the test dataset. The easiest way to evaluate the efficiency of an implicit-feedback based recommender system is by verifying if a user actually clicked the displayed recommended items. The percentage of recommended items clicked and subsequently carried until purchase provides a measurable conversion rate, i.e. click-through rate (CTR). Nevertheless, in the experiment setup, any CTR data is not available in the absence of real users *clicking* on the recommendations. Thus, an offline method is used to examine the overall accuracy of IFBRS. Spark MLlib ALS APIs [271] are used to predict future ratings. Developed in Scala, parameters are configured as shown in the Listing 1:

Listing 1: Recommendation parameters for Spark ALS

```
1 val als = new ALS()
2   .setMaxIter(5)
3   .setRegParam(0.01)
4   .setUserCol("UserID")
5   .setItemCol("MovieID")
6   .setRatingCol("Rating")
7 val alsModel = als.fit(training)
8 val predictions = alsModel.transform(test)
```

Predicted ratings by the Spark ALS are Validated with the IFBRS rankings. If any predicted ALS ratings match with the actual ratings in the test database, the corresponding records are compared with the ranking produced by the IFBRS. The method is considered successful if an item rated as *five* by ALS is ranked among the top *three* by the ranking algorithm. See Table 6.6.

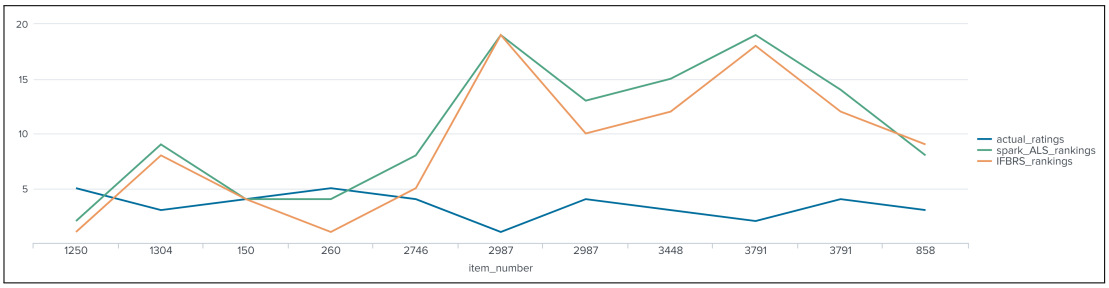


Figure 6.9: IFBRS recommendation distribution is significantly closer to the user-provided actual rating compared to the Spark ALS. The results compare favorably to IFBRS than the standard Apache Spark ALS.

6.8.6 Evaluation of Precision

The actual Click Through Rate (CTR) can not determine if a user actually clicks on the recommended items in the test environment. Therefore, an alternative method is proposed to calculate

Table 6.6: Comparing IFBRS with Spark ALS

Item number	Actual rating	Spark ALS prediction	Top n recommendation algorithm order
2987	4	4.1	10
1250	5	4.3	1
3791	4	3.9	12
858	3	2.6	9
1304	3	3.2	8
3791	2	2.5	18
2746	4	4.3	5
260	5	4.6	1
150	4	3.7	4
2987	1	2.5	19
3448	3	3.8	12

Evaluation compares Spark ALS with IFBRS rankings for accuracy. All *five* ratings are expected to rank among the top *three*.

the precision of the recommended items. User Satisfaction Index (USI) [201] is used to define precision as follows:

$$USI(i) = \frac{|A_i \bigcap B_i|}{min(|A_i|, |B_i|}) \times 100\% \tag{6.9}$$

The equation reveals how closely recommended items are related to the actual user provided ratings. In particular, if an item is ranked among the top *five* recommended items A_i, then the actual rating (B_i) is expected as *four* or *five*. Precision is defined as follows:

$$P = \frac{1}{n} \sum_{i=1}^{n} USI(i) \tag{6.10}$$

In principle, precision denotes the average satisfaction index across all the user bases [201].

Table 6.7 is based on the precision function defined in Equation 6.10. Precision accuracies in IFBRS are 2.4% greater than SVM, 8.6% greater than Logistic Regression, and more than 30% greater than all other algorithms. Therefore, in comparison with alternative models, IFBRS im-

Table 6.7: Model Prediction Accuracy for Recommended Items

Algorithm	Precision	Recall	Accuracy	F1 Score
GaussianNB	0.15	0.27	0.19	0.14
Bernoulie NB	0.42	0.45	0.44	0.51
Decision Tree Classifier	0.55	0.64	0.70	0.60
SVM	0.79	0.57	0.63	0.57
Logistic Regression	0.74	0.54	0.53	0.51
Proposed IFBRS	**0.81**	**0.82**	**0.82**	**0.81**

proves classification and regression accuracy significantly on multiple scales and parameters. See Figure 6.10.

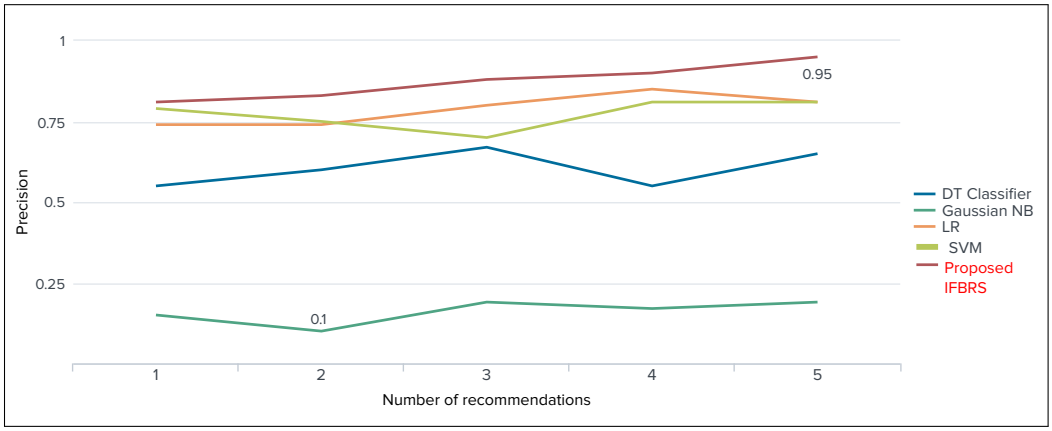


Figure 6.10: Model precision comparison against the number of recommended items.

6.8.7 Evaluation: Alternative Approach

An alternative and *more measurable* approach is presented in this section. Spark ALS ratings are converted into ranks and compared with IFBRS. The method evaluates metrics more effectively by comparing accuracy under the same scale i.e. ranking order (see Figure 6.9). Also, Root Mean Square Error (RMSE) is computed based on the ranking system under the latent factors such as location, time, and user preferences. We can verify how each factor independently and together affects the overall accuracy of recommendations. The following steps are performed for

computing RMSE: (i) The ratings are computed for each of the recommended products with Spark ALS, (ii) ratings are converted into relative ranks for each of the latent factors, (iii) RMSE are computed using user-provided actual rating and Spark ALS predicted rating. Similarly, RMSE for IFBRS rankings is obtained with each of the latent factors while keeping user-provided rating as the reference (Figure 6.11). Observe, the recommendation provided by the IFBRS improves in accuracy after applying latent factors. The accuracy is significantly improved by employing all of the factors together. Figure 6.12 shows RMSE for IFBRS batch and *lifelong learning model*. Incremental lifelong learning produces better accuracy compared to one-shot batch learning.

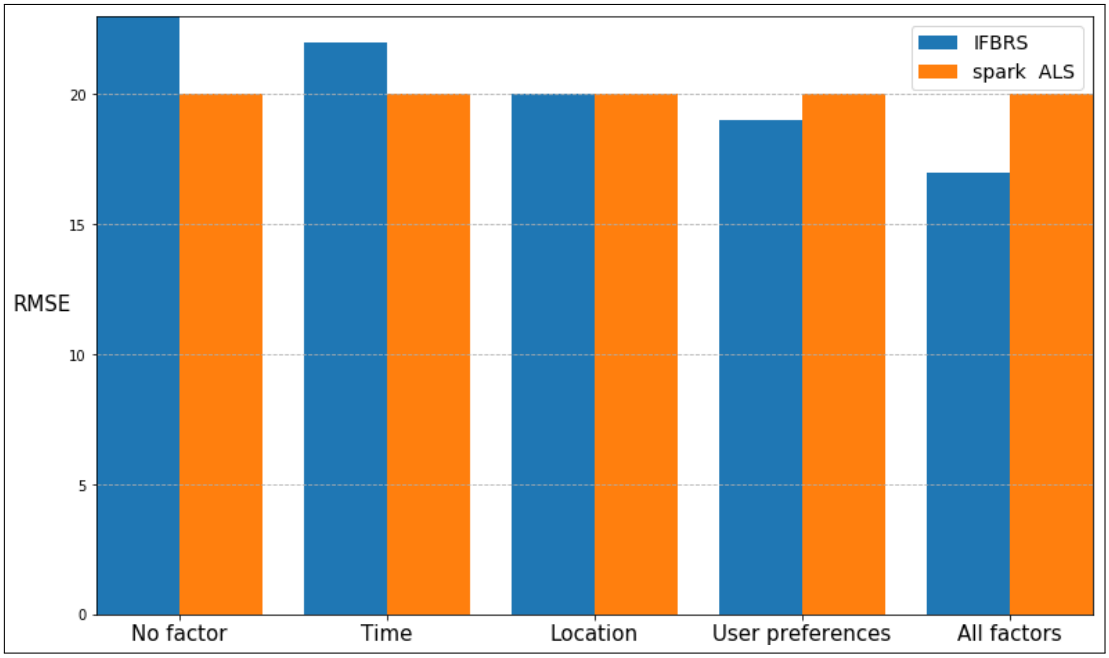


Figure 6.11: RMSE for IFBRS and Spark ALS. RMSE is computed by applying the latent factors. The best accuracy is achieved when all of the factors are put together.

6.8.8 Load Tests of MALA

This subsection provides system performance under stress. Apache Storm, the stream processor unit, and Cassandra, the storage engine of MALA are evaluated for load testing. Three million

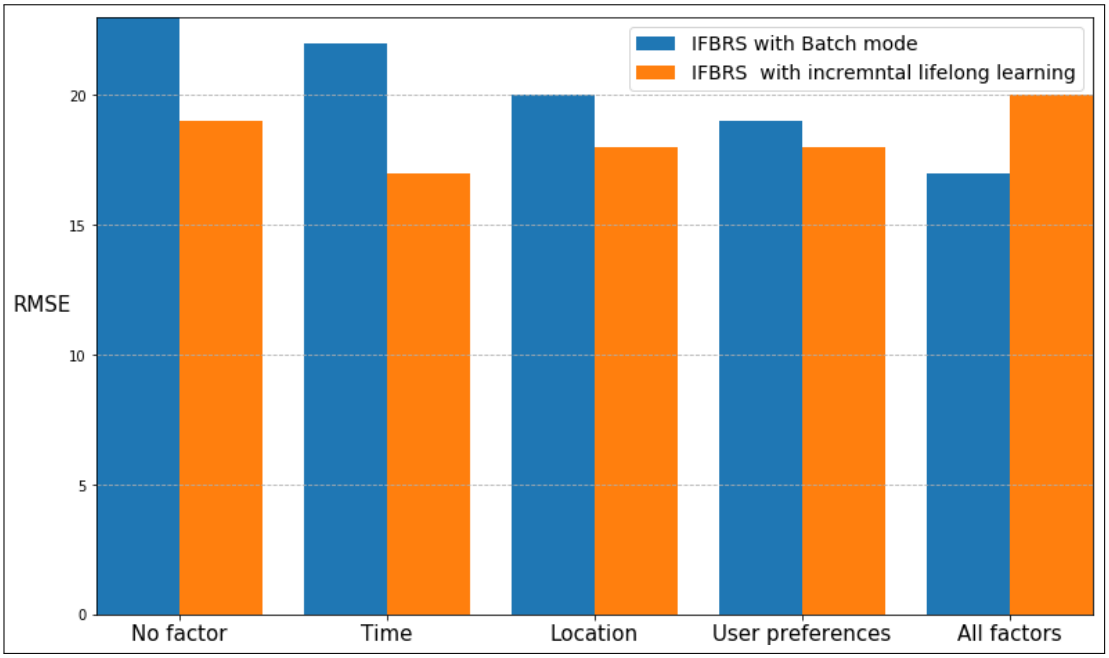


Figure 6.12: RMSE for IFBRS one-shot batch learning and *lifelong learning model*. RMSE is computed for the batch and incremental lifelong learning models of IFBRS over the latent factors. The lifelong learning model produces better results by integrating both batch and stream processing methods.

records perform the write operation first and run a mixed type of load (read plus write) for another 3 million records. The server setup is shown in table 6.8. Refer to Figure 6.13 and 6.14 for test results using Datastax OPSCenter platform. The results validate the scalability of the system under stress.

Table 6.8: Cassandra Setup

Cassandra Vendor	Number of Records	Replication Factor
DSE	6 Million writes	3

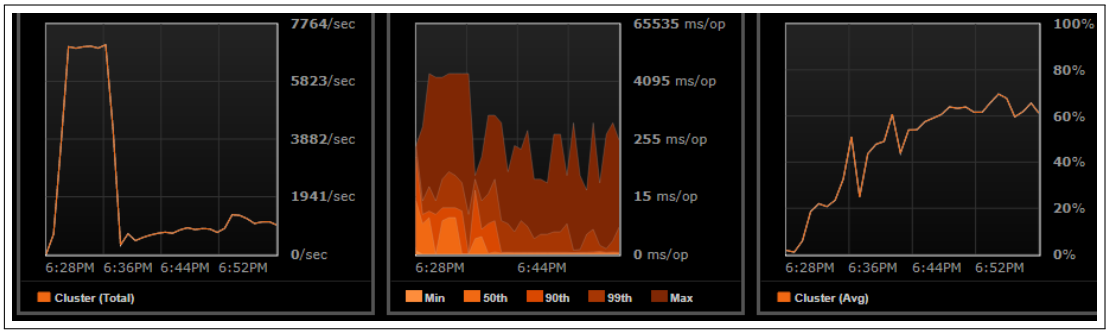


Figure 6.13: Statistics shown for: (i) Read Requests, (ii) Read Request latency and (iii) OS Disk Utilization. *(i) Read Requests:* per second read requests count on all coordinating nodes. Analyzing the number of requests for a given period reveals about the system read overhead and usage trends. *(ii) Read Request latency (Percentiles):* 99th, 90th percentiles, min, max, the median for a client reads. When a node accepts a client read request, the time period initiates, and it terminates while the node replies back to the client. Depending on the replication factor and consistency setting, this might include the network delay from the data's replicas. *(iii) OS Disk Utilization*: CPU time used by disk I/O. The time unit is in milliseconds.

6.9 Discussion

6.9.1 Comparing IFBRS with Spark ALS

The IFBRS is considerably more accurate than the Spark ALS APIs when context parameters, e.g. location and time decay function, are considered. The algorithm's approach applying higher weight

Figure 6.14: Statistics shown for: (i) OS Load, (ii) Heap Used and (iii) TP Flushes Completed. *(i) OS Load:* Operating system load average. One minute value parsed from proc/loadavg statistics on Linux systems. *(ii) Heap Used:* Average of Java heap space utilized.*(iii) TP Flushes Completed:* Number of memtables flushed to disk since the nodes start.

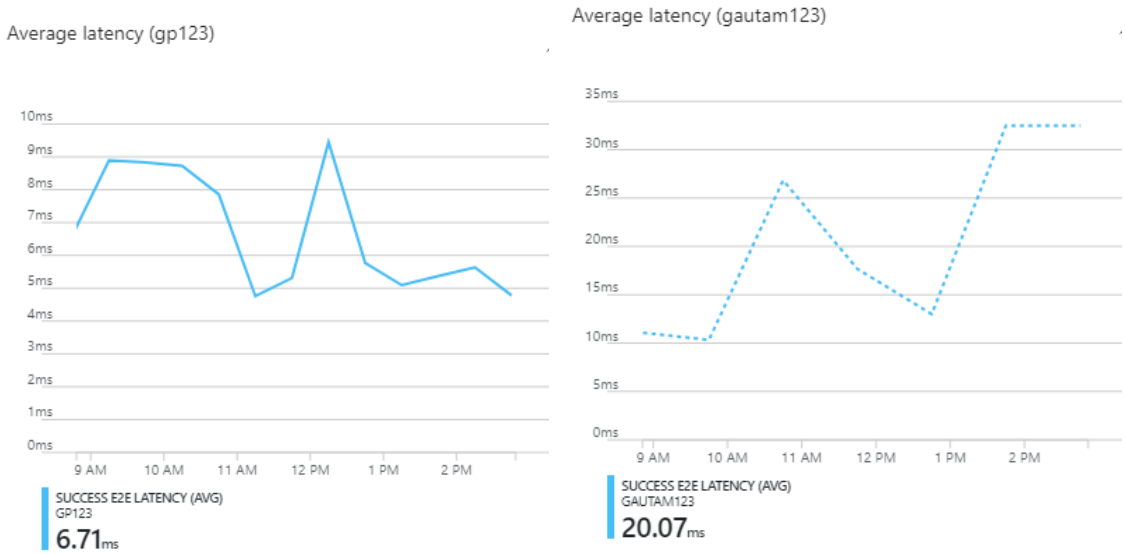


Figure 6.15: Screenshots from Microsoft Azure cluster with Storm installed. Average latency in Azure HDInsight storage unit 1 (fig a) storage unit 2 (fig b).

on recent data over historical records increases the *relevancy* of the recommendations, resulting in a higher *Click Through Rate*. Results validate the claim that the user's *context parameters* significantly improve recommendation quality in personalized recommendations.

6.10 Comparing the Efficiency of the MALA Hybrid Learning

MALA is a collaborative consolidation framework for stream and batch data. The framework's real-time and batch agents use Spark MLlib APIs. MALA offers the following distinct advantages over traditional batch and stream frameworks:

1. **Collaboration between real-time and batch agents.** Real-time and batch agents collaborate, retrain, consolidate, and exchange learning to provide deeper analytical insights.

2. **Infrastructure Cost Savings.** Computationally it is not feasible to train the entire set due to large cluster requirements. The incremental method allows the framework to be trained iteratively on a small amount of data at a relatively modest infrastructure setting with commodity hardware.

3. **Faster Training Time.** The application can not always wait until the entire dataset is collected. The iterative approach can help training the model with the available dataset.

4. **Quick Adaptability.** The trained set can adapt much faster to keep up with the new updates in the dataset by retraining iteratively. The load test results further assert the scalability of the system under stress conditions.

6.10.1 Trade-off Between Low-Latency and High-Accuracy

As depicted in Figure 5.3, the proposed MALA architecture consists of both low-latency real-time component and a high-accuracy batch counterpart, simultaneously working on the same dataset. The cost of batch and a real-time algorithm is defined by time, space, and resource utilization complexity. The competitive ratio of a streaming algorithm is computed against a batch algorithm. A

smaller than *one* cost ratio proves the superiority of the streaming process. However, the competitive ratio due to time complexity turns out to be greater than one in all scenarios and particularly worse in a certain dataset. This is less surprising as streaming algorithms process in mini-batches without the foresight of the entire dataset. On the contrary, real-time algorithms benefit from space and resource utilization complexity due to the much less volume of data processing in each mini-batches. A real-time algorithm can be called as α competitive if there are positive factors α and γ so that:

$$v_i \leq \alpha v_b + \gamma \tag{6.11}$$

v_i is the cost for the streaming algorithm
v_b is the cost at the batch setting

From Equation 6.11, we conclude, the cost of an α competitive real-time algorithm is *not* inferior than α times to a optimal batch algorithm (v_b) plus some initial advantages (γ) assigned to the batch algorithm due to its prior knowledge of the entire dataset.

In a classic trade-off between low-latency or high-accuracy, the case study using a real-time dataset selects low latency responses over *high accuracy* and *strong consistency*. The framework uses the Cassandra database to support low-latency requirements in real-time functionalities. Cassandra initiates a *read repair* to update the inconsistent data with a higher consistency setting. The client read-write processes, therefore, must wait before discrepancies are eliminated. Big Data systems serving faster turnaround time at a near-real-time requires setting the lowest possible value for consistency (which is consistency *one*) due to the negative effect of consistency levels on general responsiveness. Cassandra supports tunable consistency settings. Therefore, in the proposed framework, Cassandra can offer the CP, i.e. Consistency and Partition Tolerant or AP, i.e. Available and Partition Tolerant with regards to the CAP theorem [86].

6.10.2 Validation by Click Through Rate (CTR)

To validate the recommendation accuracy, an offline strategy is adopted where 80% of data is used for predicting the recommendation, and the remaining 20% for validation. However, the Click-Through Rate (CTR) and purchase rate (PR) are more accurate ways to verify recommendation accuracy. The CTR and PR can measure a user's preferences on viewing or buying from recommended items vs non-recommended items. As the CTR or PR verification requires a production deployment with a large set of the real user base, it was not a feasible option in the current test environment.

6.11 E-commerce Applications at the Streaming Layer

The case study with the Recommender System uses a long-running batch layer in the MALA. In that context, a list of functionalities can be performed *simultaneously* at the streaming layer that requires quick response time. Experiments use the production dataset obtained from data.world [21] and the data generation app OpsDataGen.spl [22] consists of data from Indian e-commerce major *Flipkart*. Experiments are carried out in Amazon Cloud Services (AWS). Results are based on MALA collaborative mix processing framework, using the batch and streaming linear regression APIs from the Spark MLlib libraries.

Building recently viewed product list. We assume, a user u_i successively views items *a, b, c* at the same session at time t_1, t_2, t_3 where $t_3 > t_2 > t_1$. Recently viewed products for user u_i is the reverse chronologically ordered list of products, i.e. c, b, a. The primary challenge is, identifying each user uniquely, even when they are not logged in. A context id is created for each user's click data in the absence of a user ID. User's *context* is derived from a session object, which is associated with each time user newly opens the website. Subsequently, a new session and a new context are captured. The context is a uniquely derived object created from the session object created at the JavaScript layer. Context ID is appended to each user's click data. Also, a recently viewed product list influences the batch layer outcome to compute the final order of the recommended products for

each user (discussed in the previous section).

The streaming layer brings about the following results :

- Forecast chart for network traffic load for the e-commerce portal. See Figure 6.16.
- Punch card view of user behaviour in the e-commerce portal grouped by day of the week. See Figure 6.17.
- Outlier chart for unusual buying patterns. See Figure 6.18.
- Rolling hourly prediction count of a number of checkouts. See Table 6.9.

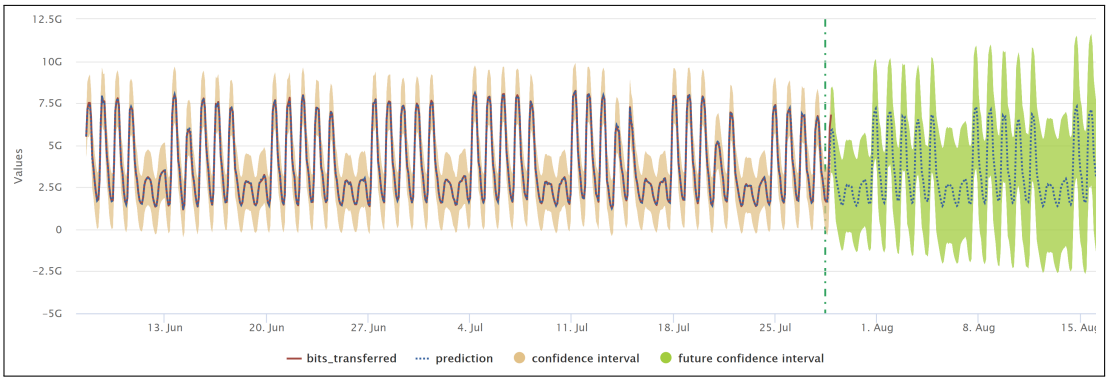


Figure 6.16: Forecast chart for network traffic load in e-commerce portal. Vertical green, dotted line differentiates between past and future data projections. Confidence interval is kept at 95%

6.11.1 Load Test

This section presents the load test results of the Cassandra database that stores and processes a real-time dataset, as described in the previous section. Server setup is shown in the table 6.10 and database setup is presented in Table 6.11. Refer to Figure 6.19 and 6.20 for test results using the Datastax OPSCenter. The results validate the scalability of the system under stress.

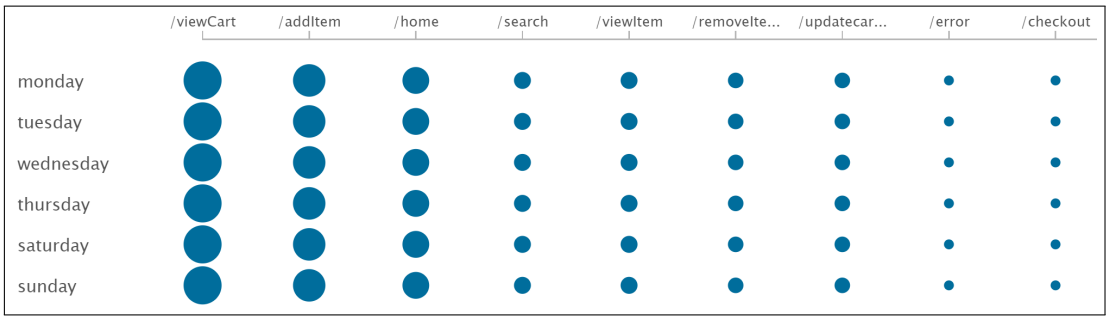


Figure 6.17: Punch card view of users' behaviour in the e-commerce portal grouped by day of the week. The chart reveals the conversion rate from the item added to the cart and final checkouts. Only a tiny percentage of cart additions is converted to checkouts. Since MALA uses a moving average of one week, the graph can be updated at the end of each week and subsequently predict future behaviour.

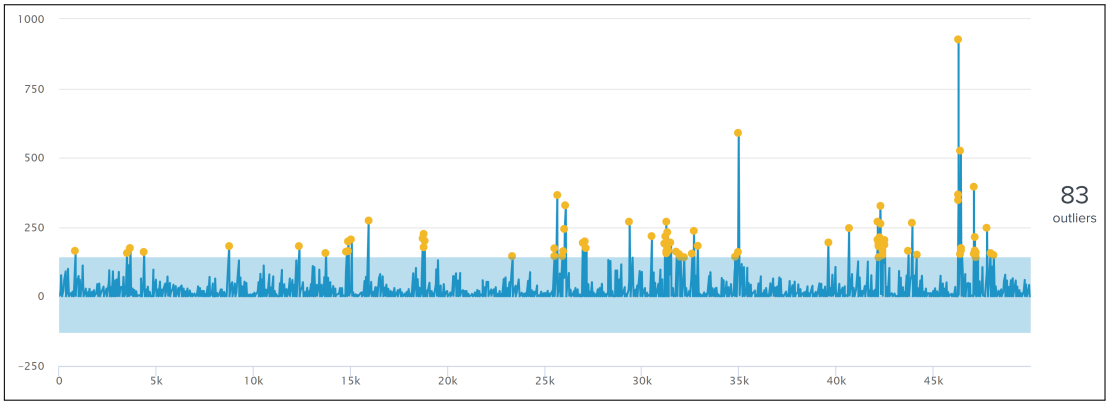


Figure 6.18: Outliers chart for unusual buying patterns. X and y axis marks the data volume and quantity of purchases. The algorithm detects the outlier of unusually high or low quantity of purchases using both stream and batch K-means clustering methods. Yellow dots mark the outliers.

time hour	actual count	lower95 prediction count	upper95 prediction count
0:00	106	105.0630306	106.9369694
1:00	208	207.0629725	208.9370275
2:00	206	205.061734	206.9357845
3:00	186	185.0629788	186.9370212
4:00	206	205.0631755	206.9372127
5:00	208	207.0636578	208.9376911
6:00	206	205.064078	206.9381085
7:00	186	185.0649051	186.9389334
8:00	206	205.064087	206.9381136
9:00	206	205.0643379	206.938363
10:00	188	187.0652488	188.9392727
11:00	224	223.0635381	224.937561
12:00	188	187.065613	188.939635
13:00	206	205.0645522	206.9385736
14:00	206	205.0646471	206.9386678
15:00	206	205.0647169	206.938737
16:00	190	189.0656084	190.9396281
17:00	204	203.0646925	204.9387117
18:00	208	207.0645059	208.9385247

Table 6.9: Moving hourly prediction count of number of checkouts. Table displaying the prediction of moving average of one hour. Confidence interval is 95%. To counter the cold start situations, MALA uses the historical batch data as initial offset for stream engine to update incrementally.

Table 6.10: AWS Instance Types

Instance Type	Instance Count	vCPUs	Memory	Instance Storage	EBS Optimized Bandwidth
m1.large	3	2	7.5 GB	500 GB	Moderate

Table 6.11: Cassandra Setup

Cassandra Vendor	Number of Records	Replication Factor
DSE	6 Million writes	3

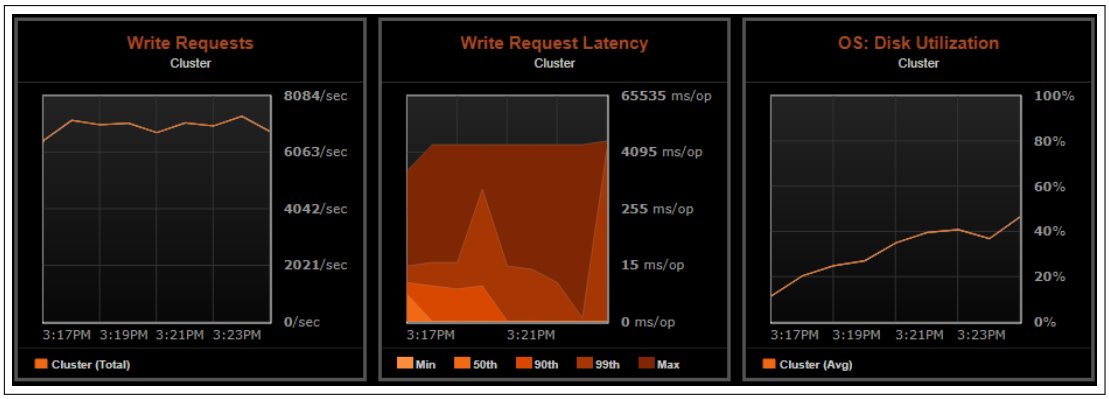


Figure 6.19: Statistics are shown for: write requests and write request latency, OS disk utilization. *Write requests:* The count of write per second on the coordinator system. Monitoring the number of requests over time exposes system write load and usage patterns. *Write request latency:* The 90th, and 99th percentiles, min, median, max of a client node write operation. The time period initiates with a node receiving a client write request and ends with the node responding back to the client. Considering the consistency setting and replication factor, this includes the network delay from writing to the replicas. *OS disk utilization:* CPU time used by disk I/O

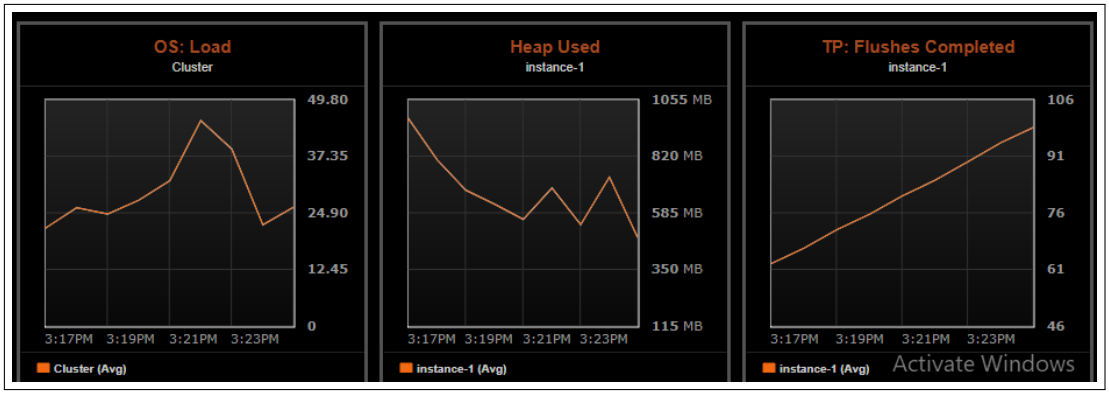


Figure 6.20: Statistics are shown for: OS load, Heap used and TP flushes completed. *OS load:* Operating system load average. One minute value parsed from /proc/loadavg on Linux systems. *Heap used:* Average amount of Java heap memory used. *TP flushes completed:* Number of memtables flushed to disk since the nodes start.

6.12 Comparing Efficiency of MALA Hybrid Learning

MALA is a collaborative, consolidation framework between the stream and batch data. The framework's real-time and batch agents use Spark MLlib APIs. MALA offers the following key advantages over standalone batch or stream only frameworks:

Smart Collaboration. Interactive real-time and batch agents can collaborate, retrain, consolidate, and exchange learning from individual domains to provide deeper analytical insights. The experimental results illustrate the significant performance improvement by low latency processing at the speed layer and providing comprehensive coverage at the batch layer. This is the key contribution of the framework and highly appealing for the big data applications looking for optimization in analytical insights.

Faster training time, quick adaptability, horizontal scalability, infrastructure cost savings. In situations where the application can not wait until the entire dataset is collected, the iterative approach can help training the model with the available dataset. The training set can adapt much faster to sync with the new updates in the raw dataset by retraining iteratively. The load test results assert the excellent scalability of the system under stress conditions. When it is not feasible to train the entire set due to large-size cluster requirements, the incremental mode supports the framework to be trained iteratively on a small amount of data at a relatively modest infrastructure setting with commodity hardware.

6.12.1 Limitations

The framework has the limitation of keeping the identical codebase in stream and batch layers that produces the same results. The stream layer often performs the additional functionalities (like recently viewed products in the case of the recommender system), but the framework still *redoes* all the major tasks by the batch layer. This leads to a typical sync problem, wherein any changes made in one layer need to be mirrored back to the other. Also, having trained on the same volume of a dataset, the accuracy of the proposed model is not any better compared to the batch only model

for certain functionalities which do not involve collaboration between two modules. A number of e-commerce analytics applications, especially those not involving large datasets, may find the architecture overly complex.

6.13 Case Study-2: Geospatial Analysis of New York Taxi trips

The proposed MALA is validated with an exploratory data analysis using a large volume New York taxi app dataset consisting of both *geo-location* and *timestamp* parameters. The case study implements the proposed hybrid data processing Multi-agent Lambda Architecture, which combines low latency real-time frameworks with high throughput Hadoop batch frameworks over a large distributed setup. The work addresses the high latency problem of MapReduce jobs by simultaneous processing at a speed layer to the requests which require quick turnaround time like real-time taxi demands, anomalous traffic detection, etc. At the same time, the batch layer in parallel provides comprehensive coverage of data by providing insights into taxi usage data.

Taxis are the mirrors of city life. The case study uncovers people's mobility patterns, explores a city's divergent landscapes, provides real-time help to taxi operators, commuters with taxi demand forecasts and anomalous traffic data. Taxi trip data tells the story of a city and its people. A few scenarios are analyzed in this subsection with geospatial, temporal data: (i) How bad is the traffic during rush hour? (ii) What is the quickest way to reach the airport? (iii) What are the city's favourite weekend destinations? (iv) How does the day of the week affect ridership? Taxi trip data uncovers it all. At the same time, taxi operators can leverage the real-time forecasting of region-wise breakups about taxi demands and efficiently load distribute. Operators and commuters benefit from real-time classification and anomalous traffic detection.

Dataset. New York Taxi & Limousine Commission (TLC) has released the city's taxi trip data from January 2009 through December 2017 [32]. Considered as a whole, the data consist of nearly 2 billion records. *Three types* of taxi data are featured in the set: the yellow, green, and For-Hire Vehicle (FHV) taxi. The yellow and green taxi trip data include records capturing pick-up and drop-off times/dates, pick-up and drop-off geo-locations, travel distances, fares, rate categories,

payment categories, and passenger counts. The dataset is more than just a vast record of pick-up and drop-off coordinates; rather, it represents an explicit story of New York city life [37] [95].

Spatio-temporal queries create intuitive visualization at the batch (Figure 6.22) and stream layer (Figures 6.21) of MALA. Prediction using the batch-stream ensemble learning is demonstrated in Figure 6.23. The incremental real-time machine The analysis uses longitude and latitude as location parameters for the year 2016 on green taxi usage.

Figure 6.21: Analyzed at the stream layer of the MALA, showing the number of pick-up and drop-off from different neighbourhoods. Taxi cab operators can track real-time taxi demands in different neighbourhoods. Figure a shows data for number of pickups and Figure *b* show number of drop-offs. Figure *c* groups the number of pickups by hour (bar chart), and an overlay line graph shows pick up demands by fare price and time of the day. Figure captures the statistics of the Green Taxi in the year 2016

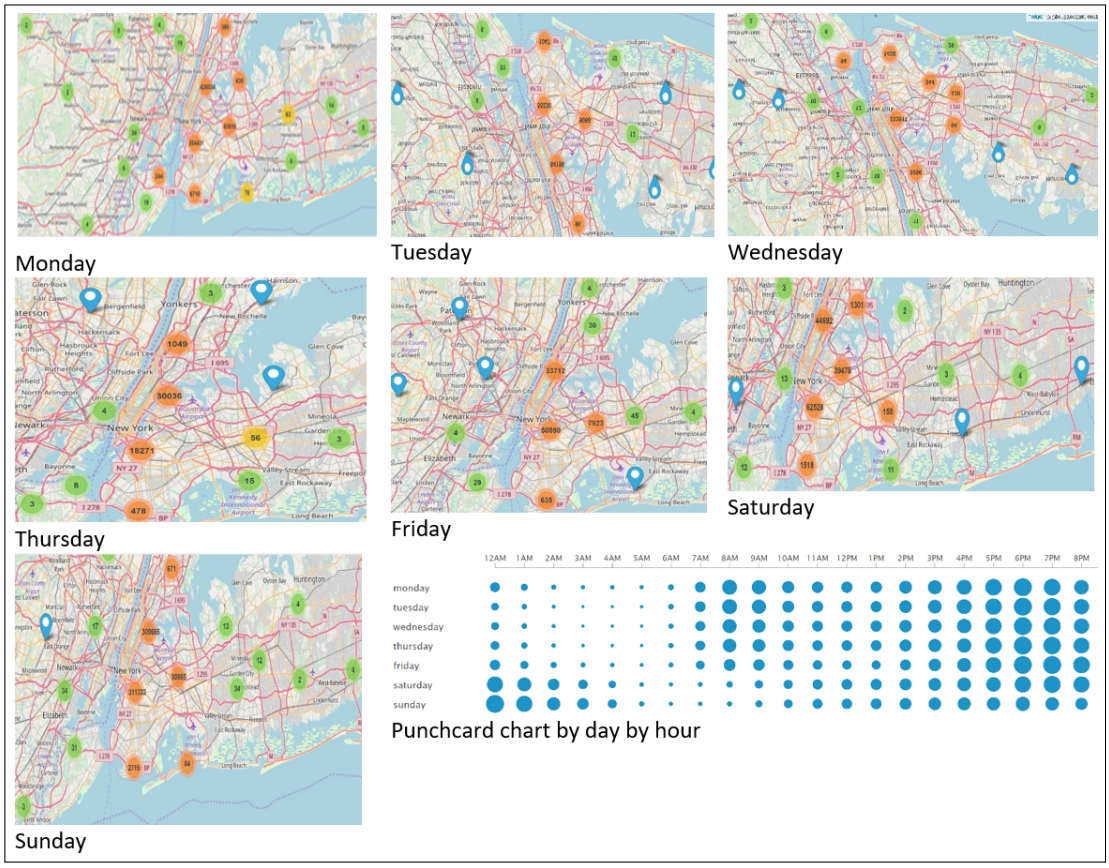


Figure 6.22: Analyzed at the batch layer of the MALA, Figures reveals pickup load to airport from different neighbourhoods grouped by day of the week in New York city. Figure in the punchcard visualization further drills down pickup demand to airport from different neighbourhoods by week of the day and hour of the day.

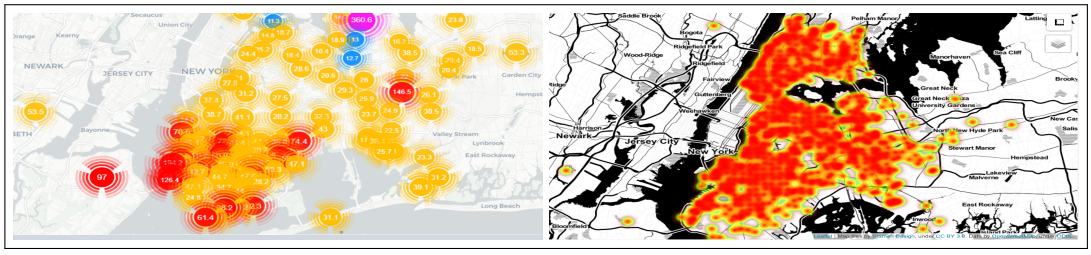


Figure 6.23: Predicting time to commute to airport (in minutes) from different neighbourhoods in the New York city. The *real-time estimate* of time to reach is useful to the operators and commuters.

6.13.1 Discussion: Using MALA to Analyze Spatio-temporal Dataset

The case study with New York Taxi app dataset demonstrates the efficiency of MALA to query both stream and batch data within a single architecture. The case study also evaluates the MALA by analyzing the Spatio-temporal dataset. The batch component of MALA unveils taxi demand to airports from different neighbourhoods for each day of the week. The streaming component analyzes the real-time taxi demand from the city's different neighbourhoods. As described in the ensemble learning in Section 6.4, the initial training model is created with historical batch data, and the model is updated incrementally and predicts continuously on each fresh arrival of real-time data. The batch and real-time integration are validated by predicting the commute time to the airport from different neighbourhoods. Therefore, MALA can be efficiently used to analyze geospatial datasets requiring both batch and real-time functionalities.

6.14 Conclusions

This chapter is an extension of the lifelong learning model introduced in the previous chapter. The chapter presented a stream-batch ensemble learning method. The Streaming model initiates with *saved learning* from the batch as an offset and is incrementally updated at the streaming setting. The framework allows the merging of a static historical data pool with the most recent streaming model. A proposed boosting based ensemble model *incrementally boosts* the accuracy on each iterating mini-batch, enabling the model to accumulate knowledge faster. In each iteration, a weak base learner is boosted as a weighted sum of previous learners. The base learners adapt faster in smaller intervals of a sliding window, improving the accuracy rate by countering the *concept drift*. The proposed frameworks provide a solution to the existing limitations of the Hadoop framework, which is inherently batch-oriented.

The proposed ensemble model leads to developing an improved version of implicit feedback based contextual recommender system through the MALA framework using a lifelong learning approach. This case study aims to build a lifelong learning system using a real-time batch hybrid

processing approach. The Architecture can use past knowledge, update and accumulate information iteratively over a large volume of data pool through a host of big data tools and methods. Integration between stream and historical batch data provides deeper insights at low latency. The proposed *Implicit Feedback Based Recommender System* used *contextual parameters* and *weighted hybridization* strategy to mix historical batch and near-real-time datasets and can predict recommendations accurately. The trained model continuously improves over time with an incremental lifelong learning method. Importantly, the model does not rely on the explicit rating provided by the user; rather, it computes a user's passive endorsements by click data. The method did not suffer from the sparsity problem of rating-based user similarity approaches. The recommendation algorithm did not force the user to log in to provide recommendations and is capable of providing accurate recommendations for non-logged-in users. While the batch layer serves the recommendation engine, the stream layer simultaneously performs real-time operations such as forecasting network traffic load, buying patterns, and moving average of predicted sell etc.

Finally, the proposed framework is verified with the New York taxi app dataset consisting of a geospatial, temporal big dataset. Taxi apps generate a large volume of real-time streams, as well as the dataset consists of past Years' historical records. MALA can efficiently analyze data to uncover the city's mobility pattern.

Advantages of the proposed model over Hadoop MR and Spark ML APIs are in terms of improved machine learning accuracy, faster response and training time, managing cold start situations, and significant infrastructure cost savings. The limitation includes inherent design complexity, sync issues, and functional redundancies of the two layers.

Chapter 7

Multimodal Integration of Text, Visuals and Metadata

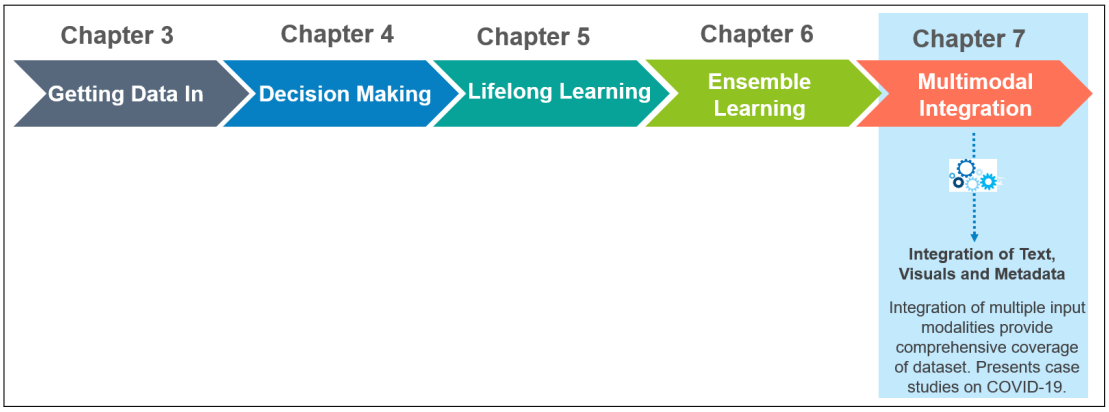


Figure 7.1: An outline of Chapter 7 and end-to-end data flow pipeline through different chapters.

7.1 Introduction

Data analysis methods described in previous chapters were limited within data input from *textual* sources. However, real-world data scenarios are full of multi-sensory data that humans perceive in

the form of reading, visualising, and hearing. For instance, images are often related to some text explanation; text contains images focusing on an idea in an article. Each modality is characterised by its own statistical features. For instance, images are often represented by pixel intensities, while texts are represented by discrete vectors embedding. Multiple modalities intertwined along with language, metadata, and visual cues, can increase prediction accuracy compared to traditional mono-modal systems using text-only data. To this end, this chapter presents an *Ensemble Learning method* by integrating up to *three* input data modalities: language, visuals (image and video) along with textual metadata for comprehensive coverage of a dataset and provides a more realistic real-world representation. Therefore, as part of the proposed Multimodal Analytics Framework (MAF), this concluding chapter of the thesis enables MALA to integrate non-textual input along with textual content across real-time and batch formats described in Chapters 3, 5 and 6. More specific research questions and the contributions are laid out in Section 8.1.

Multimodal analysis traditionally involves conceptualising abstract frameworks for language, images, and other resources and their intersemiotic relations (e.g., text and image relations) and then demonstrating these frameworks with a small or medium-sized dataset. This scenario has changed with the recent move towards multimodal approaches to big data analytics, which involves empirically testing and validating multimodal theory and frameworks by analysing large data sets. However, large training sets of analysed texts are required to develop computational models based on new multimodal methods. Therefore, an alternative approach that involves integrating multimodal frameworks with state of the art computational models for big data, cloud computing, natural language processing, image processing, video processing, and contextual metadata is proposed. The integration of these disparate fields has the potential to improve computational tools and techniques dramatically, thus placing multimodality at the forefront, the research aimed at mapping and understanding multimodal communication. As a step forward in this direction, this chapter also explores how computational tools and approaches can be integrated into the *Multimodal Analysis Platform (MAP)* as a practical implementation of the proposed *Multimodal Analytics Framework (MAF)* with facilities for searching, storing, and analysing text, images, and videos in online media,

along with dashboards for visualising the results.

7.2 Current Research Gaps

This section highlights: (a) the benefits and current limitations of the big data approach to multimodal analysis, and (b) the need to incorporate knowledge about language, visuals, metadata, and other resources as semiotic systems (rather than simply sets of symbols and pixels) to improve computational techniques for big data analytics.

Multimodal learning and prediction involve the embedding of multi-sensory data that humans perceive in the form of reading, visualising and hearing. Despite recent advancements in machine learning and big data technologies, software models significantly fall behind the manual understanding of cognitive tasks involving the aggregation of text, image, and video. Consequently, in reality, the state-of-the-art computer models are still *monomodal* specialists, leaving the scope for humans to logically interpret joint modalities in aggregation. Collecting and analysing a multimodal dataset in a meaningful way based on a single analytical platform remained a major challenge. In this case, the primary objective is to create a joint interpretation of multiple knowledge representations and create a unified view that produces a more accurate and comprehensive representation of data than a single standalone knowledge base. Next-generation artificial intelligence prediction systems need to intercept such multimodal signals to save lives by producing early warning signals ahead of time. Personalised medicine, for instance, is customised for each patient based on multimodal information, including genotype and phenotype parameters, to achieve a therapeutic response. However, forecasting the impacts of a pandemic spread solely on social media data sources is a relatively new research area.

Crawling the web in real-time to extract a large volume of text, image, and video data quickly and accurately and index them to make data immediately searchable for creating interactive analytical reports requires merging a divergent set of technologies. Meaningful analytical insights require that the data collection process takes into account both historical and real-time datasets based on different filters (e.g. date range, users, keyword searches etc.). That is, the current ap-

proach of collecting *everything* within only a limited time range can not authentically represent data as it contains a large amount *noise* of unrelated records, e.g., web-scraping data that need to be filtered out. To develop comprehensive insights on a range of topics, multiple *sub-topics* need to be aggregated together along with processing and interaction of different modalities like text, image, video, historical batch data, and real-time data stream. In addition, data can be harvested from multiple channels simultaneously (e.g. online news media, public reports etc.) to provide further contextual information and metadata.

7.3 Summary of Contributions

This chapter aims to introduce new machine learning methods to join multiple modalities in addition to stream and batch introduced in previous chapters and undertake prediction tasks to address the aforementioned research gaps. The **first part** of the chapter introduces new methods for multimodal classification, Topic Modeling, entity correlation, and clustering to train and predict based on multiple modalities between language, visuals (i.e. language, images and videos) along with metadata. Then the chapter describes the MAP that implements the proposed MAF in an end-to-end multimodal analytics solution, data generation to reporting, on big social and news media data. The MAP provides the optimal fusion of disparate streams from multiple modalities and data sources (in the current case study, Twitter, newspapers and public health reports). MAP minimises the manual interpretations of the combined modalities through the provision of: (i) efficient information extraction, ingestion, and cleansing process; (ii) effective integration of data from social and news media in a cross-media interaction scheme; (iii) leveraging the latest technologies in data science, big data and Natural Language Processing (NLP); and (iv) easy-to-use intuitive web-based interface, making technological advancements in multimodal analysis accessible to the non-data scientists (social scientists, linguists, etc.).

Using the proposed methods, the **second part** of the chapter presents *three* case studies that are the most contemporary and relevant topic related to the COVID-19 dataset. The *first case study* is about the prediction of public health and social issues as the COVID-19 situation unfolds.

The proposed system provides early indications of the impact of pandemic spread, such as new illnesses, recoveries, deaths, and job losses, by analysing social media conversations and validating the predictions with published reports. Followed by a *second case study* on *COVID-19* illustrates the data growth dynamics and prediction of information diffusion in media using the MAP. A *third case study* explores how social and news media response towards COVID-19 swings before and after George Floyd's death in Minneapolis, the USA, on 25 May 2020. Results suggest that multiple modalities with language, metadata, and visual cues can increase prediction accuracy compared to traditional mono-modal systems using text-only data.

The research attempts to answer the following high-level research questions:

RQ1. How can be *three* modalities: text, visuals (image and video), and textual metadata be integrated on top of the real-time and batch collaborative models described in preceding chapters for language analysis tasks such as multimodal classification, Topic Modeling, entity correlation, sentiment classification, and clustering?

The chapter also deals with the following research questions through the case studies:

RQ2. What is the multimodal response function that describes social media response in relation to a current event and scientifically provides early indications of COVID-19 spread such as new illnesses, recoveries, deaths, and job losses?

RQ3. What is the multimodal response function that describes social media reactions in relation to published news events that guide the trend dynamics in social media?

RQ4. How to predict the persistence or decay of social media trends with respect to the volume of additional news articles posted on a similar topic?

The specific aim of the research is as follows:

1. To provide a joint interpretation of multiple knowledge representations in terms of linguistics, visuals, and metadata across real-time and batch data presented in preceding chapters and create a unified view that produces a more accurate and comprehensive representation of data compared to a single standalone knowledge base.

2. To introduce new methods for multimodal classification, Topic Modeling, Entity Correlation,

and clustering to train and predict based on multiple modalities between language, visuals (image and video), along with metadata.

3. To develop deep semantic interpretation algorithms for cognitive computational vision based on the new innovations for image, text, video, their relations and context analysis.

4. To introduce new machine learning methods to join multiple modalities and carry out prediction tasks and an evidence-based assessment on the same.

5. To develop a cloud-based Multimodal Analytics Platform that facilitates end-to-end solutions, from data generation to reporting, on big social and news media data.

6. Validate the model using case studies on *COVID-19* based on social and news media data.

7. To develop an integrated semi-automated open and extensible prototype multimodal system for contextual analysis of the text, image, and video relations in online communications.

8. To develop an easy-to-use web-based interface to collect social and news media data by web searches, storing data in JSON format, and indexing them for faster search performance.

9. To develop a case study to define multimodal response function that describes social media reactions in relation to current events and provides early indications of COVID-19 spread such as new illnesses, recoveries, deaths, and job losses.

10. To establish a multimodal response function that describes social media response in relation to published news events that guide the trend dynamics in social media.

11. To predict persistence and decay of social media trends with respect to the volume of additional news articles posted on a similar topic

7.4 Theoretical Framework: Proposed Multimodal Predictive Model

The thesis improved several machine learning results using the batch-stream collaborative model and MALA framework such as Expectation-Maximization (EM) algorithm with Gaussian Mixture Model (GMM) in Chapter 3, clustering, Random Decision Forest Regressor and Classifier and sentiment classifier in Chapter 5, Collaborative Filtering in Chapter 6. This chapter extends the previously introduced methods along with *three* additional modalities as language, visuals (image and video) and metadata. This section introduces *two* new methods for machine learning using: (i) multimodal classification and (ii) multimodal topic modelling, entity correlation and clustering (iii) multimodal sentiment analysis. The multimodal classification uses the ensemble method between Convolutional Neural Network (CNN) for visual features, Long Short-Term Memory (LSTM) for textual features, and embedding of numerical features. An alternative approach to LSTM is provided for the multimodal classification using Gated Recurrent Units (GRU), Support Vector Classifier (SVC) or Stochastic Gradient Descent (SGD). In the multimodal Topic Modelling approach, a method is proposed to detect topics using Named Entity Recognition, Parts of Speech and establishing correlations among co-occurring elements (see details below). The following state-of-the-art algorithms provide a topic modelling approach for clustering similar topics: Latent Dirichlet Allocation (LDA), Truncated Singular Value Decomposition (SVD) and Non-negative Matrix Factorisation (NMF) methods. Sentiment classification uses Valence Aware Dictionary and sEntiment Reasoner (VADER).

7.4.1 Multimodal Classification Proposal

A joint predictive analytics method is developed by the multimodal integration of (i) textual features modelled by the LSTM, (ii) visual features modelled by CNN, and (iii) numerical features such as sentiment polarity, the subjectivity of the text, and word length. Using the integrated model, a dataset is classified into more than 35 categories as document classifications. Within each of the categories, predicted and actual results are compared for accuracy. The method is described in detail as follows.

Word embedding creates a dimensional reduced numerical dense vector representation of each word term, taking into consideration of *sequence* as they appear in the corpus and the *context*. To generate the embedding, raw text is cleansed by removing the stop words, Stemming and Lemmatization. The proposed MAP (described in the next section) provides an easy-to-use web-based interface to filter out unwanted records by Splunk Search Processing Language (SPL) [54]. SPL allows users to construct complex data filtering criteria based on date range, personalities, places, or topics to create relevant *vocabulary* for the embedding. Words in the vocabulary are then converted to numerical feature vectors. Vector embeddings are created by the *Spacy* libraries with training data from Kaggle [23, 238]. Each trained embedding is mapped to a word in the vocabularies and subsequently encoded by LSTM, a class of Recurrent Neural Network. In the next step of the processing pipeline, LSTM is converted to a dense vector. The dense vector is added with the context vector from the *attention* [155, 266]. The consolidated vector is used for classifying the text into one of the classification labels for news based on headlines and short descriptions as shown in Table 7.1.

Table 7.1: 35 Classification Labels (Document Level)

Classification Labels
arts & culture, black voices, business, college, comedy, crime, education, entertainment, fifty, good news, green, healthy living, impact, Latino voices, media, parents, politics, queer voices, religion, science, sports, style, taste, tech, the worldpost, travel, weird news, women, world news, world post, parenting, home & living, environment, weddings, food and drink, money, wellness.

Similar to the language component, visual features (images and videos) are converted into linguistic word annotations using the *Clarifai* (https://www.clarifai.com/), which is an external service to MAP. Linguistic descriptions extracted from visual elements pass through similar steps as text to classify the image/video into one of the classification labels from Table 7.1. Classification outcomes from textual and visual features produce a *confidence score* for each of the classification labels. Also, the user selects appropriate weights for the language and visual features from the interface (See Figure 7.6. The confidence score for each classification label is multiplied with the

user-selected weight for text and associated visuals. Language and visual scores are added together to predict a new classification label. The process is described in Figure 7.2 and 7.3

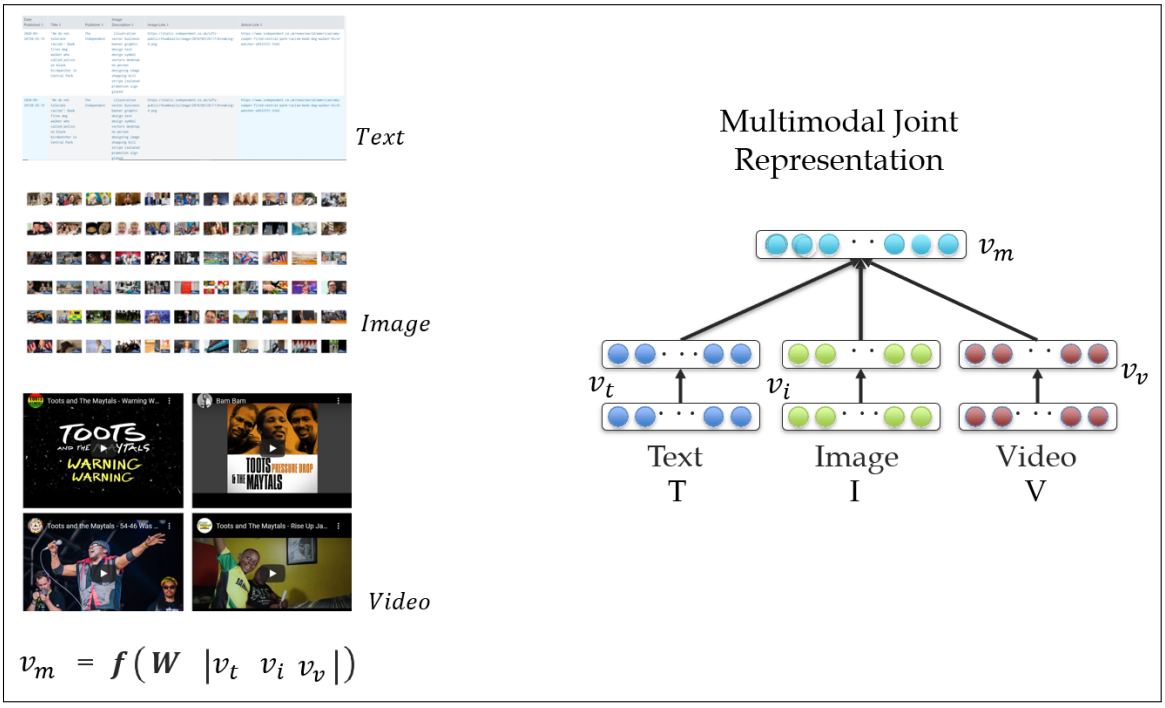


Figure 7.2: Multimodal integration between text, image and video. Each dataset are converted into numerical feature vectors before joint representation. Weight (W) is applied to each vector.

The multimodal embedding process is explained using the image (Figure 7.4) associated with a news article. The image is converted to textual descriptions by the *Clarifai* image classifiers, with the following results: *administration, people, business leader, politician, man, chair, home, football, portrait, meeting, democracy, league, leadership, banking, intelligence, flag, parliament, military achievement.* MAP classifies the news text and associated images into one of the 38 classification labels. The full list of classification labels is displayed in Table 7.1. Text and image classification results are combined in accordance with the confidence scores, and individual weights are applied to the image and text. After applying weights on each modality, the joint score is computed by combining text with image modality. Users can select the weight scores for text and images (see Figure 7.6). The confidence score is the probability value of an image or text

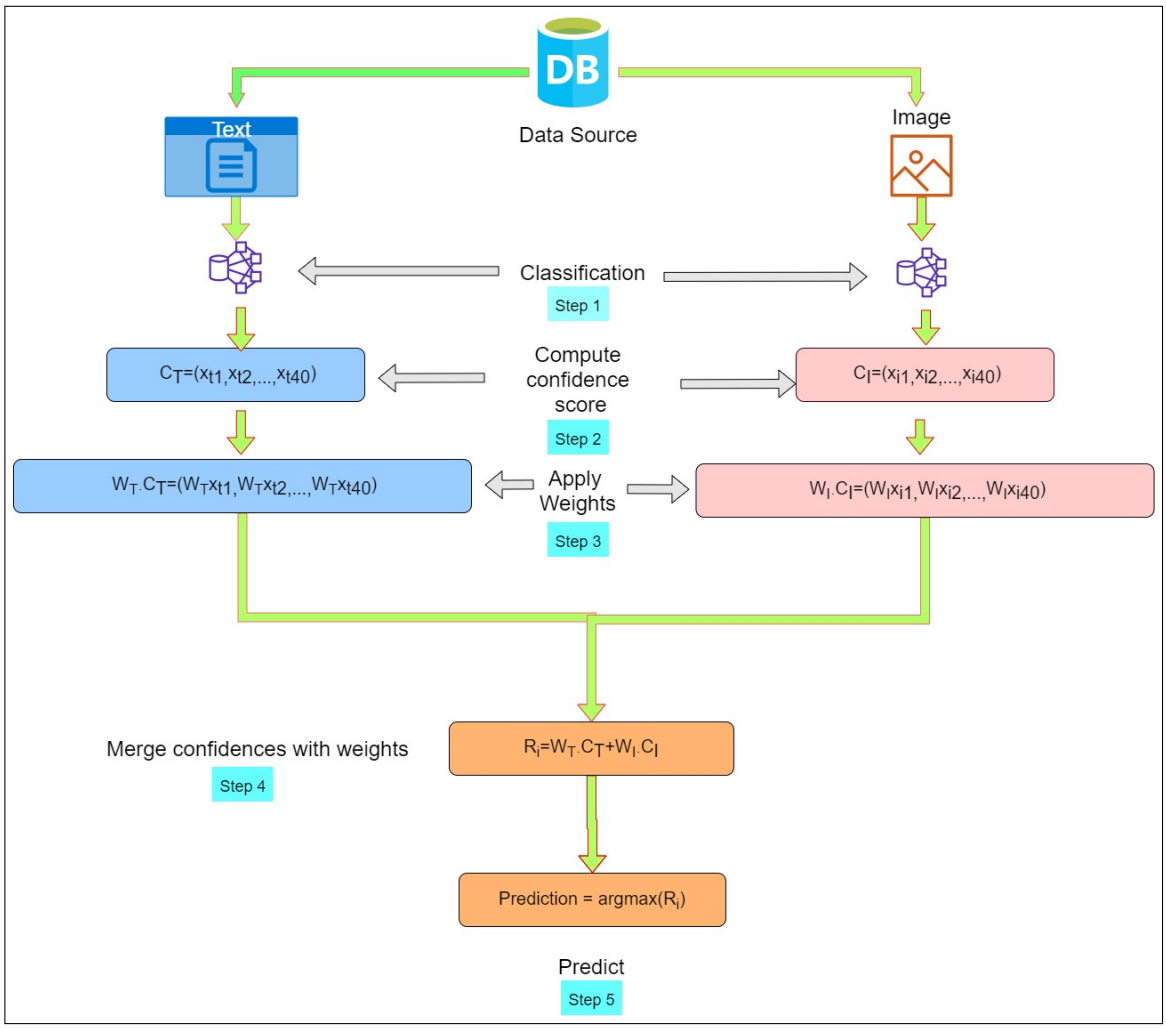


Figure 7.3: Flowchart for multimodal classification model. Text and images are classified separately using Python Keras libraries (**step 1**). User selects a classification algorithm from the available options: SVC, LSTM, GRU or LSTM with metadata (discussed below). Confidence values are computed for each classification labels in Table 7.1 (**step 2**). User selected weight is multiplied with confidence score (**step 3**). Text and image results are added together (**step 4**). The argmax function selects a classification label with the highest prediction value (step 5).

being classified into a particular label. For example, text in the article is classified as *black voices* with a probability of 30%, and the associated image descriptions are classified as *politics* with a probability of 70%. As equal weights are applied for text and image (50% each), the combined classification label is computed as *politics* (see Figure 7.7).

Figure 7.4: Image associated with a news article converted to textual descriptions. MAP introduces a new model of multimodal fusion between language (text), visuals (image), and textual metadata. MAP supports joint intermodal representation across a large number of images and associated texts.

The integration of the language and image results influence the overall classification outcomes. For example, we can compare the classification results of newspaper articles with the keyword *Coronavirus* for the week before George Floyd died [1] on (a) text only (see Figure 5), and (b) text and image results displayed (see Figure 7.5). The classification results differ, with a decrease in the number of articles classified as politics (from 29.6% to 19.0%) and an increase in the number of

[1]George Floyd was a 46-year old black man who was assassinated during an arrest by four Minneapolis police officers on 5 May 2020 [182]

articles classified as travel (from 4.6% to 12.5%), sports (7.0% to 8.7%), and entertainment (7.3% to 7.7%). The results demonstrate that the inclusion of image data (as textual descriptors) significantly influences the classification outcome, suggesting that the integration of text and image modalities has the potential to increase the understanding of the meanings made in online communications. The larger implications of these findings are evident, namely that it is imperative to develop robust methodologies for image analysis in order to understand discourse patterns.

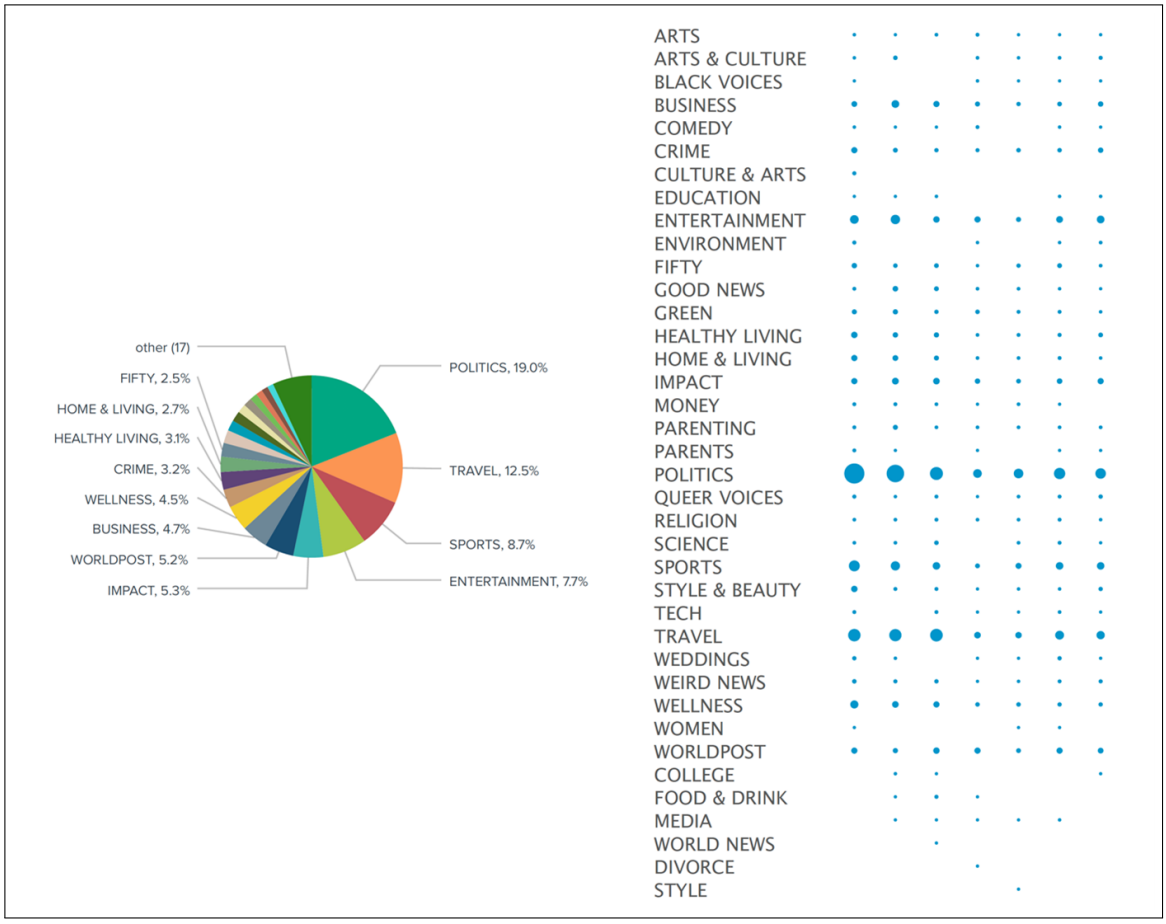


Figure 7.5: Multimodal classifications of Newspaper articles (text and image) with keyword *Coronavirus* for Week Before George Floyd death.

Consider Figure 7.7, text and associated image are classified as *Black Voices* and *Politics* respectively. The classification outcome is based on images of Donald Trump and Joe Biden, as well

as the associated news headline and news summary. Assume a confidence score for text and images is 0.4 and 0.8, respectively. Since equal weights are applied to the textual and visual component, after combining text and image, a combined classification label is computed as *Politics*.

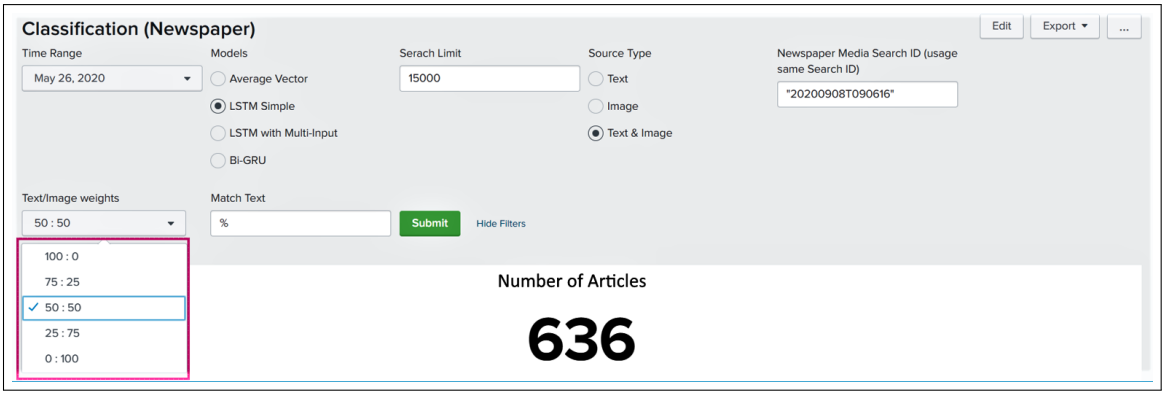


Figure 7.6: Multimodal Classification user interface. User can select text and image weights.

Date	feature_img	image	title	Summary	Text_Predict	Image_Predict	Predict
2020-05-26	https://static.independent.co.uk/s3fs-public/thumbnails/image/2020/05/25/19/trump-biden-mask.jpg	administration people business leader politician man chair home football portrait meeting democracy league leadership banking intelligence flag parliament military achievement	As a black man who's watched white Republicans fake outrage over Biden saying 'You ain't black', I need you to know this	Nonetheless, the way Biden worded the comments hurt some black voters, who have carried the Democratic Party for decades and handed Biden his primary wins this year. Many of the reactions to Biden's comments were measured, advocating that the former VP should do more to assure the black community he isn't taking them for granted. Some used this as an opportunity to increase pressure on Biden to pick a black woman as his running mate. If you're black and reading this, this is why your vote should not be deterred by Biden's comments. President Trump smeared the first black president as being born in Kenya, spent his presidency trying to undo his legacy, and is now leading calls to imprison him.	BLACK VOICES	POLITICS	POLITICS

Figure 7.7: Final classification label is predicted as a aggregation of text and image results in accordance with user selection

The multimodal ensemble process is further enhanced with an additional modality as *textual metadata* which considers *three* types of numerical features: *sentiment polarity, subjectivity,* and *word count*. Sentiment polarity can range between -1.0 (extremely negative) to +1.0 (extremely

positive) while subjectivity score is a float that spans between 0.0 (very objective) and 1.0 (very subjective). Subjectivity expresses *feeling* in a document, while objectivity indicates a *fact* as detected by Python Scikit-learn libraries. Word count is the number of total words in each document. Metadata are converted to a numerical feature vector and integrated into the text embedding. Numerical features of metadata are integrated with LSTM as an *early fusion* at the vector level (see Figure 7.9). Combined output from text and metadata is added with results from the visual content as described above.

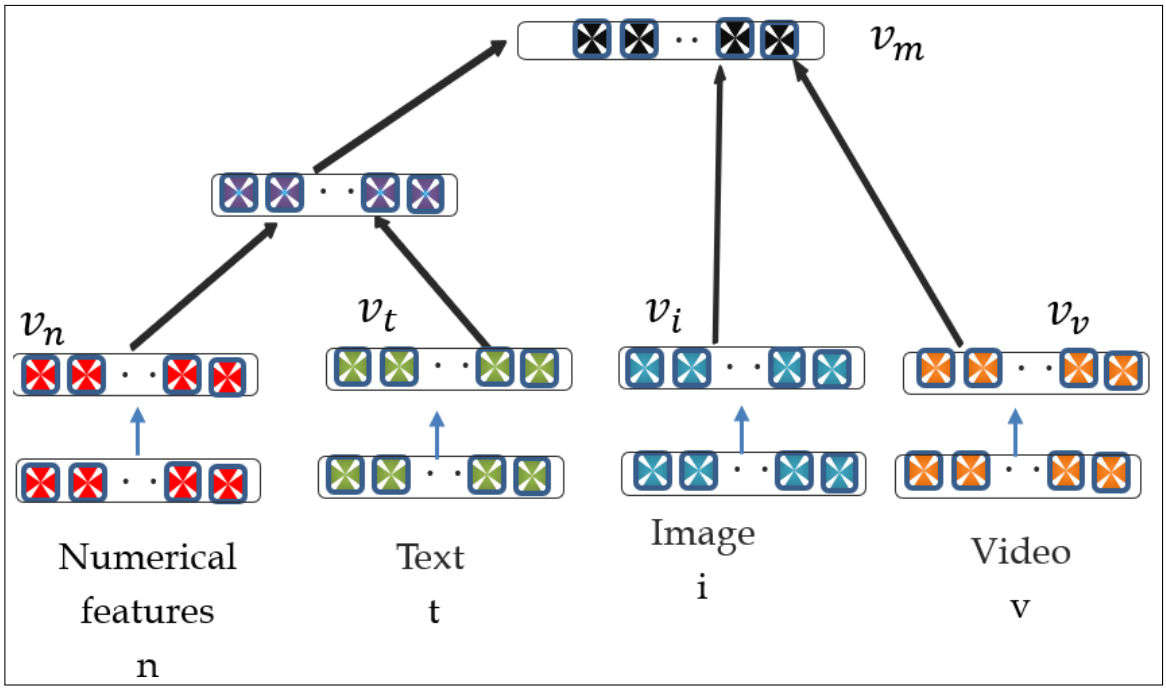


Figure 7.8: **Multimodal integration of *three* modalities.** Language features are decoded using Long Short-Term memory (LSTM), visual features are encoded with CNN. Numeral features such as sentiment polarity, subjectivity and word count are extracted from text. A simple multilayer perceptron integrates modalities.

Recurrent Neural Network model with LSTM is compared with *two* alternative approaches: (a) GRU and (b) SVC/SGD that does not use RNN. The GRU Units provide a possible substitute for LSTM in the Recurrent Neural Network implementation. GRUs can be trained rapidly than LSTM, are simpler to implement without a *memory* layer, and perform well for less amount of

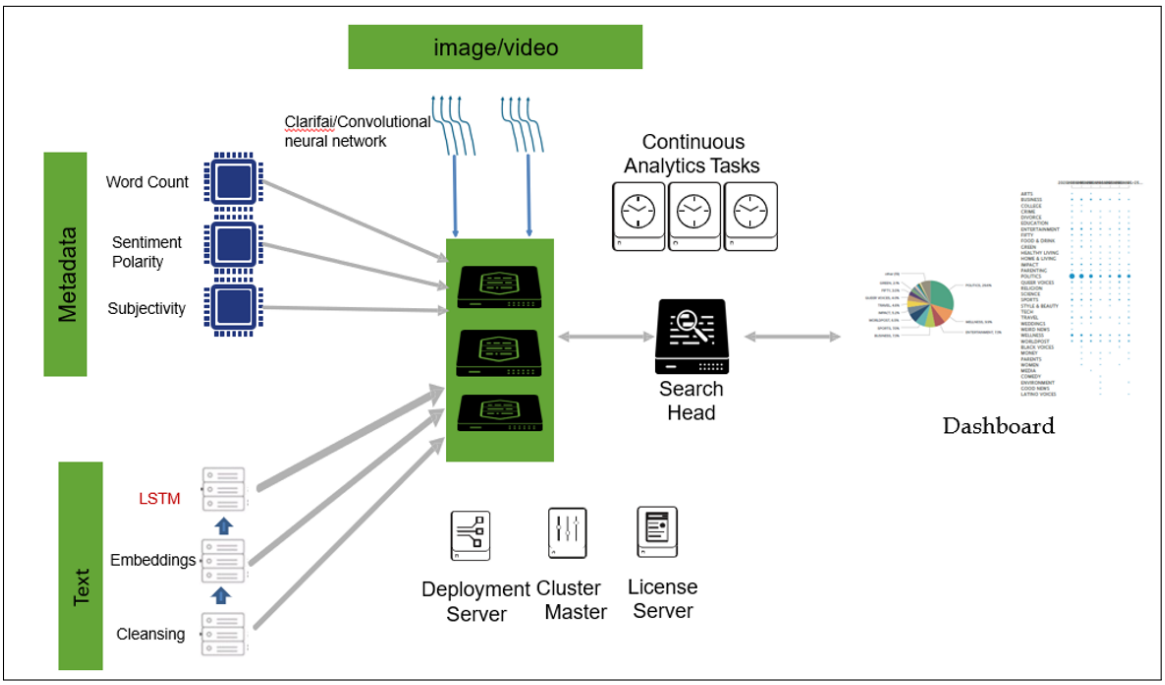


Figure 7.9: **Multimodal integration in cloud environment.** Deployment Server, Cluster Master and License Server are running within Splunk Software. Splunk also provides a search head where continuous analytics tasks are run by users to combine multiple modalities and create reports.

data [68, 264]. Therefore, we can recourse to an alternative method by replacing the LSTM layer with *two* GRUs. The *second alternative approach* with SVC/SGD does not use RNN and word embedding. With this approach, data is first cleansed by stop-words and SQL queries for unwanted records. The filtered data is then converted to a Term Frequency-Inverse Document Frequency (TF-IDF) vector [203, 272]. The final multimodal data analytics pipeline merges the TF-IDF vector is with (a) metadata: subjectivity, sentiment polarity, and word count; and (b) image descriptions by CNN image classifiers.

7.4.2 Multimodal Topic Modeling, Entity Correlation and Clustering

In traditional analytical systems, semantic interpretation in relation to the context is lost as the analytical systems break sentences into words. A random collection of standalone words in isolation cannot analyse the corpus in accordance with every theme or topic. Therefore, it is essential to tag each term to its Named Entity, Parts of Speech, and establish meaningful connections by the term co-occurrence to understand the context in which words are used. MAP adopts an advanced multimodal fusion scheme that integrates textual and visual features based on the context and themes of the corpus with an objective to derive insights on the following *five* questions: *who* is the person speaking and when? *What* is the topic he is speaking about and *where*? *How* does he feel about the topic? Why did he feel that way? To achieve this, first, we can convert the visual features to textual descriptions using *Clarifai* image classifiers or CNN. Image descriptions are combined with associated texts as the *simple union* between two modalities as defined by Equation 7.1. The joint representation is extracted for the Parts of Speech (POS), Named Entity Recognition (NER) and establish correlations among the POS and NER tags such as who, what, when, why, etc. Fine-grained language analysis by the MAP recognises as many as *19* Named Entity and *37* Parts of Speech on both textual and visual features (see Table 7.2).

Parts of Speech or Named Entity tags that appears within a *single record* are considered *co-occurring* to each other. Co-occurring tags are ordered based on their relative frequency within the corpus, and the connections between tags are represented by the Ribbon chart (see Figure 7.16).

Table 7.2: List of Parts-of-speech and Named Entity tags that MAP recognizes.

Parts of speech	**Named Entity**
Coordinating conjunction	Cardinal: Numerals that do not fall under another type
Cardinal number	Date
Determiner	Event
Existential there	FAC: Buildings, airports, highways, bridges etc.
Foreign word	GPE : Countries, cities, states
Preposition or subordinating conjunction	Language
Adjective	Law
Adjective, comparative	Location
Adjective, superlative	Money
List item marker	NORP: Nationalities or religious or political groups
Modal	Ordinal: "first", "second", etc
Noun, singular or mass	Organisation
Noun, plural	Percent
Proper noun, singular	Person
Proper noun, plural	Product
Predeterminer	Quantity
Possessive ending	Time
Personal pronoun	Work_of_art
Possessive pronoun	
Adverb	
Adverb, comparative	
Adverb, superlative	
Particle	
Symbol	
To	
Dollar	
Interjection	
Verb, base form	
Verb, past tense	
Verb, gerund or present participle	
Verb, past participle	
Verb, non-third person singular present	
Verb, third-person singular present	
Determiner	
Pronoun	
Adverb	

The joint modality is established through the union of language and visual tags for each record as given by the Equation 7.1:

$$d^{combined} = [(W_i^{text})|W_i \in d^{who/what/when/where/why}] \cup [(W_j^{image/video})|W_j \in d^{who/what/where/when}] \tag{7.1}$$

An alternative approach to Named Entity or Parts of Speech is proposed for Topic Modelling with any of the following algorithms: LDA, SVD, or NMF to create clusters of words with similar topics. Each record is first tokenised, and stop words are eliminated. Terms are then *lemmatized*, *stemmed* [2], and a *TF-IDF* vector is created. Topics are modelled using TF-IDF vector from the one of the available methods to choose from i.e. *LDA*, *SVD* or *NMF*. Words in the corpus are clustered according to the topic by selecting any of the available clustering algorithms: *KMeans, Birch, Spectral Clustering, DBScan, and XMean*. While each algorithm may produce slightly different results, when clustering into an unknown number of clusters, DBScan is more useful than others.

7.4.3 Multimodal Sentiment Analysis

Sentiment Analysis uses VADER to compute sentiment for each word that considers the context and semantics of sentences [114]. VADER returns sentiment scores between -1 and 1 for each term, corresponding to positivity, negativity, and objectivity (neutrality). From which a compound score is computed as the overall score for each term. Aggregation of the compound sentiment scores of the lexical features (e.g., words) provides the sentiment score for each record. VADER does not require any training data but is developed using a rule-based, human-constructed, gold standard sentiment dictionary. The approach aims to leverage the low-resourced, rule-based model to develop a multimodal sentiment analysis to integrate visual and linguistic sentiments. The principal technique for the *visual sentiment analysis* revolves around modelling and identifying the senti-

[2] Stemming and Lemmatization both returns the base form of inflectional form of words (such as reading to read). Stemming may not generate meaningful dictionary words, but lemma is an actual language word

ment expressed through gesture, facial expression, and background. While gesture identification or image captioning is already a well-trodden research field, sentiment extraction of non-verbal expressions is a relatively less explored research area [236]. Similar to multimodal classification or topic modelling, NLP and computer vision-based emotion recognition involves interpreting multimodal inputs into a monomodal dataset by converting visuals into linguistic descriptions. Subsequently, the VADER is used to identify the sentiment polarity for each modality separately and integrate them. The method is described as follows:

Step-1: Extract descriptive tags, i.e., concepts for images and videos, by Clarifai's Predict API.

Step-2: Compute the compound sentiment scores from textual content and the associated visual components (images and videos).

Step-3: A-weighted hybridisation strategy combines the sentiments relating to user-provided weights on the language and visual components.

Step 4: Repeat step-2 and step-3 for each record in a dataset to compute the aggregated sentiment on the entire corpora using Equation 7.2. Aggregated sentiment score is the average of the compound scores of all records in the dataset.

$$v_m = \frac{\sum_{r=1}^{n} f(v_t^r w_t, v_i^r w_i, v_v^r w_v)}{n} \tag{7.2}$$

Where v_m is the aggregated sentiment score, w_t, w_i, w_v are the text, image and video weightages respectively, v_t^r, v_i^r and v_v^r are the sentiment scores of text, image and video for the r^{th} record, and n is the total number of records. The user interface for multimodal sentiment analysis is similar to multimodal classification as illustrated through Figures 7.6 and 7.7.

7.4.4 Video Analysis

As described in the preceding sections, text-visual integration requires visual features (images and videos) are converted into linguistic word annotations using the *Clarifai API*. Video to text conversion involves several intermediate steps before converting a video into textual annotations. To start with, a video is split into series of frames based on a number of non-identical scenes.

Image to text conversion is explained in previous sections. This section describes how different scenes are detected within a video. Based on the detection of each new scene, each video frame is image-detected to describe the scene with word tags. Also, a timeline for video is created, and rectangular bounding boxes are drawn on each frame of a video, distinguishing the target object from the image background. *You Only Look Once* (YOLO) object detection algorithm splits a video into a number of scenes that are semantically different and detects visual information for each *frame* [206]. Subsequently, frames are annotated into word tags using the *Clarifai API*. The workflow of video analysis is shown in Figure 7.10.

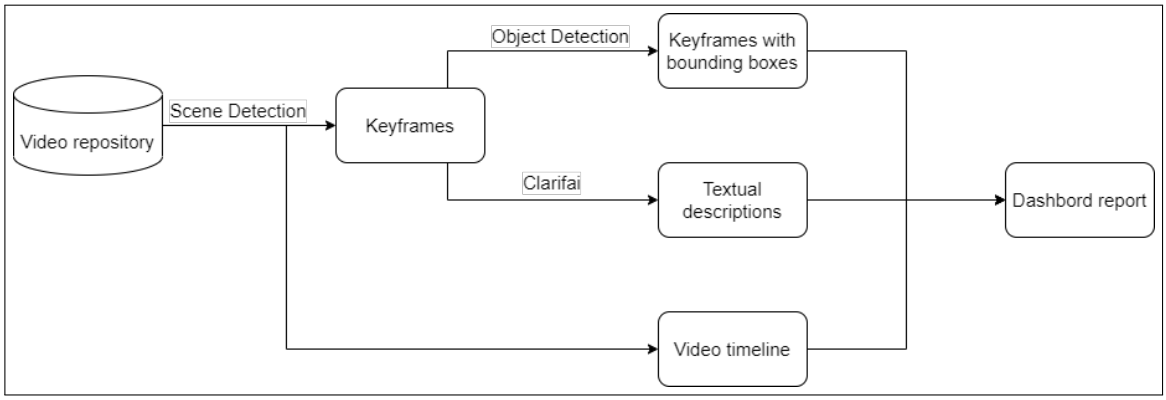


Figure 7.10: The workflow of video analysis

Scene Change Detection

Scene Change Detection algorithm compares the difference in visual content between adjacent frames against a set threshold/score denoting how different frames are, which if exceeded, triggers a scene cut.

Scene change detection generates a video timeline and also plays a crucial role in efficiently utilising scarce computing resources and processing time for compute-intensive video analysis. A scene is defined as a section of a motion picture in a single location, and a continuous video stream is made up of a series of shots (see figure 7.11). Series of frames extracted from a scene are generally similar since objects within the consecutive frames are largely unchanging in short

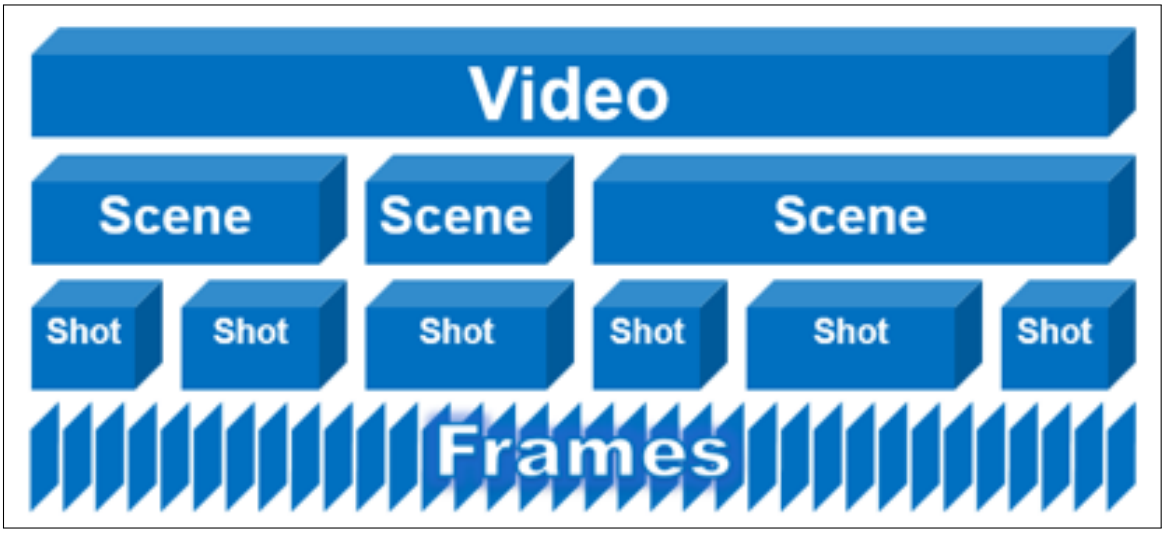


Figure 7.11: The inherent hierarchical structure of a video.

intervals. Therefore, the first frame and the last frame of a scene can be picked as the *keyframes* that are representative of a scene. All the *keyframes* are reserved for object detection in the next step, while other redundant frames are discarded. After scene detection is completed, the video is reduced from thousands of repeating frames to several *keyframes* which still retain the main features of the video. With this approach, instead of running object detection throughout the entire video, only a limited number of *keyframes* are analysed, limiting a significant computing overhead.

Yolo Object Detection

After the *keyframes* are extracted from the video, bounding boxes are drawn on the images with the help of *Yolo v3*. *Yolo* uses a single neural network to directly input the entire image for model training [206]. Since the intermediate steps of generating candidate regions are eliminated, the model can quickly distinguish the target from the image background so that the real-time detection of the target object is realised.

The *Yolo v3* network divides the input image into $s \times s$ grids of square size. Within each grid, if the centre of the target falls into a grid, then the grid is responsible for predicting multiple

bounding boxes which contain the target. Each bounding box consists of *five* predictions: *x, y, w, h* and *a* confidence score. The (x, y) coordinates represent the centre of the box relative to the bounds of the grid cell. The width and height (*w, h*) are predicted relative to the whole image. The non-max suppression algorithm is applied in *Yolo v3* network after obtaining multiple bounding boxes [268]. The algorithm removes the bounding box with a lower confidence score and keeps the bounding box with a higher confidence score as the detection box of the target. Based on the COCO dataset [30], *Yolo v3* can detect 80 COCO object classes (see Table 7.3). The class labels (word tags) and their corresponding confidence score can be displayed along with the bounding box on the image (see Figure 7.12).

Table 7.3: 80 COCO Object Classes

80 COCO object classes
person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat traffic light, fire hydrant, stop sign, parking meter, bench cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket bottle, wine glass, cup, fork, knife, spoon, bowl banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

7.5 Multimodal Analytics Platform

Multimodal Analytics Framework (MAF) is the overall end-to-end theoretical framework consisting of components such as Multi-criteria Decision Making (MCDM) and a Multi-agent Lambda Architecture (MALA). The MAF is implemented in an easy-to-use web-based platform in a public cloud called Multimodal Analytics Platform (MAP). The platform can be accessed using the following URL: http://35.246.69.64:8000/ (user name: uol, password: liverpool123). This section introduces a cloud-hosted analytics platform MAP that implements multimodal methods described in the preceding section. MAP is developed as a potential solution to the limitations of the existing

Figure 7.12: Yolo detects people, cars and traffic lights.

tools due to the heterogeneity of data and their multimodality. MAP is an analytics platform for real-time data collection, big data analytics, and interactive reporting. The key features of MAP are as follows:

1. **Social and News Media Data Generation: Collecting Text, Image and Video Data at Scale.** MAP consists of real-time data collection facilities from Twitter and five leading UK newspapers *The Guardian, The Independent, The Evening Standard, The Metro, and The Sun*. For Twitter, the user explicitly collects data through a real-time Twitter search using a keyword or user profile name. Data from news media, on the contrary, are generated at a scheduled interval by an automated process from a list of newspapers.

2. **Indexing and Semantic Annotation:** The data collection process generates a number of JSON formatted files that are subsequently indexed to improve search performance at least *10x* faster than would otherwise be achieved. Image and video data are semantically annotated using the image and video analysis tools.

3. **Search and Interactive Reports:** Multiple search results are combined at the data collection step through an easy-to-use web-based interface. The aggregation is applied across Twitter and multiple newspapers for text, image, and video by clicking on the appropriate options in the interface. Any unrelated data is filtered using the advanced Structured Query Language (SQL) and a date range. The resultant charts and tables can be exported in PDF, CSV, JSON, or XML formats.

MAP is hosted on the public cloud (Google Cloud Platform Compute Engine instance for this research). The news media trends are tracked in social media for impact analysis of one media platform on another. As shown in Figure 7.13, the raw stream is first loaded into the flat files. In the downstream processing pipeline, data moves to the Splunk indexer after cleansing. Splunk is subsequently used for search and to create dashboards.

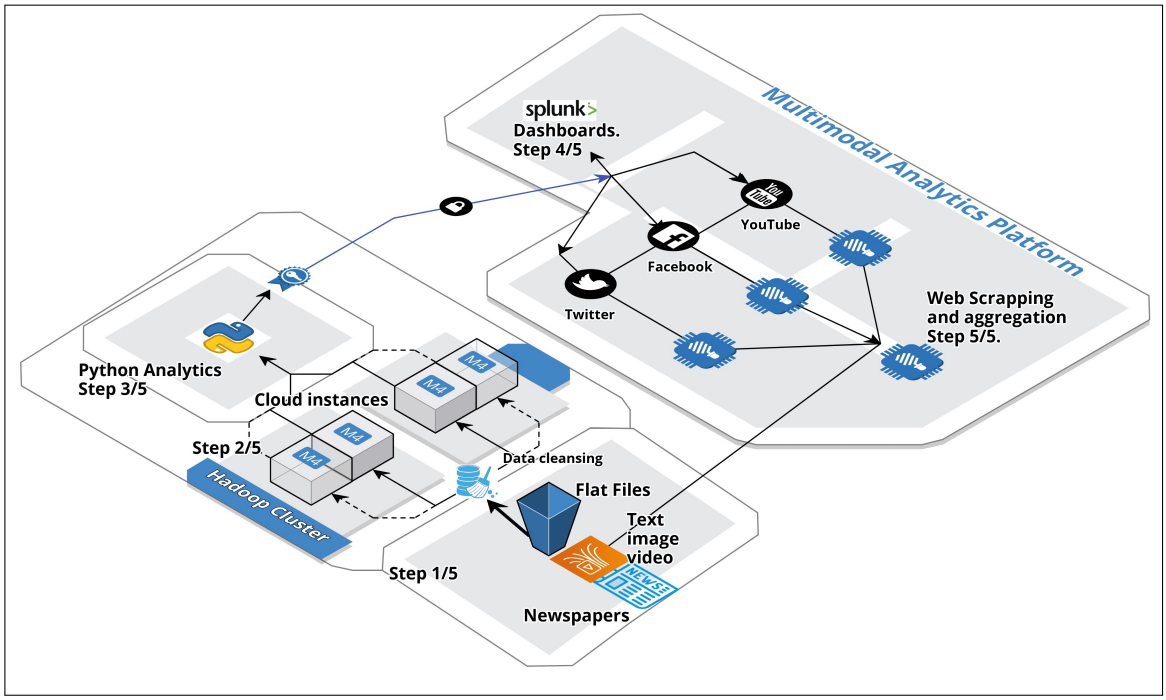


Figure 7.13: MAP deployment architecture into the Cloud Platform.

7.5.1 Information Discovery and Data Collection from Social and News Media

Data ingestion is as important as the analytics itself. MAP lets users search Twitter in real-time through an easy-to-use web interface. In the data discovery phase, users' *keyword* search filters all possible *matching Tweets* in the Twitter database. Users can refine the search query using a date range and optional location filter. In the subsequent data collection phase, *Python BeautifulSoup* [180, 180] APIs *scrapes* the Twitter pages and saves data into local file system as JSON formatted data. Each Tweet contains the meta fields along with actual Tweets, including the following metrics: Tweet text, Hashtags, Retweets, Tweet User Name, Tweet, Image and video URLs, number of likes, replies, and retweets. Also, each record contains two other meta fields indicating if the Tweet is a reply to *another Tweet* or the Tweet is an original post. Each Tweet contains a *primary key* field as *Tweet ID* that uniquely refers to a Tweet post.

Contrary to the manual searches with Twitter data generation, *newspaper* data generation is an automated backend process. A Python process scheduled through the *Linux Cron job* incrementally updates the database with all the news articles published on that day from a list of newspapers. The newspaper data collection process is programmed once in 24-hour intervals. Each news article contains the following fields: article title, date of publication, publisher/name of the newspaper (*The Guardian, The Independent*, etc.), author, image URLs, Article URL, keywords, video URLs, article summary, and the full text. Similar to Twitter, newspaper scrapping uses *Python BeautifulSoup* APIs to identify HTML elements (like title, author, etc.) from each of the article URLs.

The success of news and social media relies on instantly capturing users' interest through various images and videos. MAP uncomplicates the process to track media data (images and videos) for Twitter and newspapers. A joint representation of text, image, video is highly conducive for impactful case studies. In this regard, MAP offers a robust cross-modal text-image-video retrieval system. However, MAP does *not* download and semantically annotate media data during the data discovery and collection phase described above. Downloading millions of images and video data for all the Tweets and news articles is unnecessary and economically unviable in the cloud infrastructure using its disk and internet bandwidth. Furthermore, meaningful reports are based on

filtered datasets of relevant topics and date ranges. Hence, the proposed MAP extracts images, videos and annotates them through a media analysis tool using ad-hoc user requests. Within the MAP's media generation interface, users have complete control over the context and subject of the media that are retrieved. Users can refine media search criteria based on complex date range, name of one or multiple newspapers, a count of media searched and analysed, and can apply a joint embedding of the texts, images, and videos with just a few clicks on the buttons provided in the interface. Figure 7.14 shows a screenshot of the proposed Multimodal Analytics Platform.

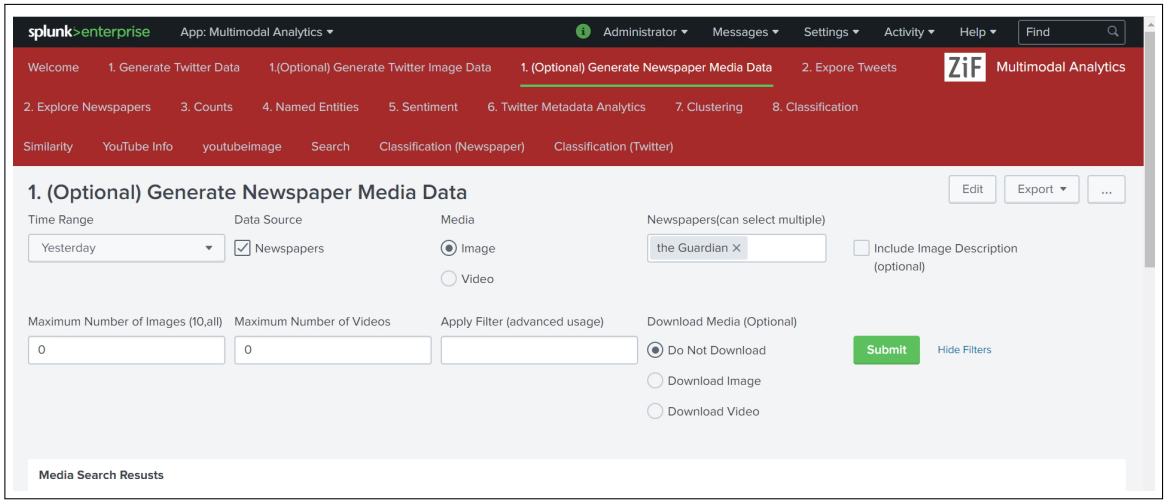


Figure 7.14: Screenshot of the proposed Multimodal Analytics Platform, showing a single dashboard and menus. The platform contains different 18 menus. The displayed page in the screenshot extracts images and videos from the newspapers and annotates them by converting images/videos into textual descriptions. The user is able to apply complex query and date range to filter the required data.

7.5.2 Indexing and Semantic Annotation

This chapter primarily addresses the multimodal correspondences between texts, images, and videos. In general, the method is based on studying the latent parameters of these media resources by annotations describing the media and projecting the embedding space between words and associated media. MAP's interactive dashboard feature lets users choose from possible combinations

in the user interface to compare and build interactive reports. One of the key characteristics of MAP is that it computes NLP and machine learning tasks *10x* faster than usual to create a useful interactive dashboard experience due to its indexing capability of each field of a record. Each record processed in MAP is time-series data, i.e. data attached to a timestamp. The timestamp field for news articles and Tweets are the *date of publication* and the date when a Tweet is posted. The MAP breaks the events based on the timestamp and ingests data into the Splunk database to index the collected data. The most recent and frequently accessed data is also stored in a cache for even faster response. Therefore, in MAP, searching through 50,000 news articles, downloading and creating thumbnails of 50 images can be completed within 10 seconds, for example.

Images and videos resulting from users' explicit searches are sent to *Clarifai* computer vision APIs [123, 137]. *Clarifai* is an external service to MAP that uses Convolutional Neural Networks (CNN) [132], a class of deep neural networks to analyse and identify media and annotate them. The automated tagging for each image or video generates metadata describing the media in a set of English words. MAP annotates thousands of images and videos into textual descriptions filtered on a time range, personality (e.g., a well-known celebrity), headline, or location. Aggregation of all the annotations along with associated Tweets and news article texts provides a 360-degree media monitoring and comprehensive coverage of the spread of information across news and social media platforms. Video recognition algorithms are used to tag the videos, grouped according to a time unit (e.g., second, minute, or hour). The *confidence score* [136, 162] ranges between 0 and 1, denoting the level of certainty attached to the description of the image or video. The higher the score, the greater *confidence* of accuracy in the result, with a score of 1 indicating the highest level of certainty.

7.5.3 Search and Interactive Report

The research aims to construct an embedding space where text, images, and videos can be combined. The resultant data can be jointly analysed through NLP techniques like n-grams, parts of speech, lemmatisation, sentiment classification, similarity, etc., and computer vision tools. The

core functionalities of the MAP include the aggregation of multiple Twitter searches, news articles or COVID related websites and the removal of unwanted noisy data. Existing social media analysis tools provide insights that simply fail to aggregate temporal modalities between social media and news media. Therefore, MAP sets out to develop an embedded space to combine the modalities between not just text, image, and video but also across social and news media, batch data, and real-time data with multiple time-series properties. The text, image, and video data collection and integration pipeline in the cloud are illustrated in Figure 7.15.

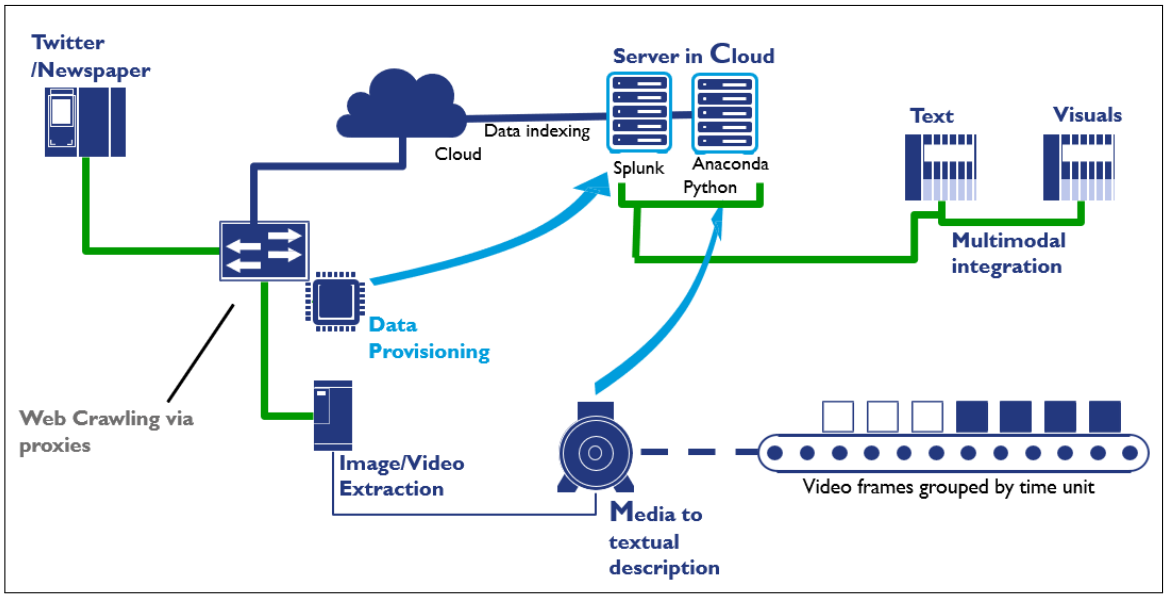


Figure 7.15: Social and news media data is collected through web scrapping. Image and video data are converted into textual descriptions and moved to the cloud environment for integration by Splunk and Python.

7.6 Case Studies

The proposed multimodal methods are validated using *three* case studies on a COVID-19 dataset in social and news media:

(ii) **Case study 1**. A multimodal approach to predict emerging situations during the COVID-19

pandemic

(i) **Case study 2**. Persistence and decay of trends: dynamics of news and social media as COVID-19 emerged and spreads.

(iii) **Case study 3.** A multimodal approach to analysing COVID-19 situations before and after George Floyd's death.

7.6.1 Motivations

Case study 1. During the COVID-19 outbreak, social media has been widely used for the democratisation of knowledge: to spread both information and disinformation, to look for recreation and humour, to stay alert and at the same time find ways of distraction from distress via the internet memes. As people relied less often on the news from traditional sources and internet searches favouring social media reliance, it resulted in the first true *social media infodemic*. Consequently, social media held the treasure trove of information repositories dictating the true nature of the pandemic spread at the grass-root level (albeit significant spread of fake news) that is otherwise remained untapped through the lens of conventional news media sources and government statistics. Non-pharmaceutical interception, like taming the power of social media, eventually becomes decisive to track and contain the evolution of COVID-19. This calls for a potential research question to develop a predictive, early indicative system about almost every socio-political sphere influenced by the outbreak. The proposed MAP combines multiple data streams to predict future eventualities such as new confirmed cases, deaths, recoveries, and job losses. Prediction is entirely based on social media conversations on Twitter. The accuracy of the proposed model is validated with the published news reporting and official statistics *1 to 4* weeks following the prediction.

Case study 2. For most of 2020, COVID-19 has taken over both social media and news media coverage. In comparison, the intense public interest in events like *George Floyd's death* died down within one month. Also, content like *Black Lives Matter* dominated news headlines in conventional newspaper sources. Recurrent retweets or shares subsequently amplify these original sources on

Twitter and Facebook. The initial spurt of trends on *George Floyd* faded away with time while persistent topics like COVID-19 continued to dominate media spaces. Nevertheless, again, the fresh incidence of black discrimination in parts of the world reinvigorated past events to become trending topics again. The persistent dominance of COVID-19 biases trend assessment with the traditional tools, since the existence of *trend peaks* makes it challenging to track *decaying* characteristics of previous short-term trends. MAP eliminates intrinsic *noise* using Structured Query Language (SQL) and Search Processing Language (SPL), which is capable of realistically separating events. From here, it is possible to undertake case studies to identify the dynamics of growth, decay, and re-emergence of trending topics during COVID-19 pandemic situations.

Social media such as Twitter are considered democratic platforms where everyone can publish, and everyone's voice is heard, which otherwise was limited to elite mass media. Mass media outlets are regarded as inclined towards their own point of view [76, 154]. The alternative opinion expressed by social media provides scope for balancing effect, neutralising slanted, authoritative media voices. Although news media effectively creates a *echo chamber* effect within the social media reactions, people selectively amplify or supplement their own belief inside a closed system that they choose to create, insulating them from rebuttal. Digital platforms can be easily tuned to feed in with the kind of information favourable to their prevailing ideology.

Do mainstream news media influences shape the social media trend? If so, can that be quantified and predicted? Identifying the impact of mass media is challenging due to its intrinsic noise, short length, and randomness in the dataset. At the time of writing, Twitter posts are just limited to 280 characters, with only 12% of Tweets posted are exceeding 140 characters. Tweeter *Microposts* are more often written in response to another Tweet or a current event that makes it harder to establish the connection between a message post and the context. Each Tweet can be attached up to 4 photos, one animated GIF, or one video. Therefore, devising a scheme for joint representation of lexical and visual modalities is essential to analyse social media impulses. Although there have been some recent successes in modelling *inter-modal* and *intra-modal* dynamics, the text-image-video embedding task is not explored to its fullest potential despite the advancements of NLP and

cognitive vision.

An answer to the aforementioned research gaps requires a new multimodal analytical system capable of collecting data from both mainstream news media and Twitter posts (text, image, and video) over time. Most significantly, the system needs to meaningfully integrate language and visual modalities, discarding the irrelevant noisy data. Pairwise comparison of trend dynamics, i.e. growth, persistence, and decay of trends on time-series data between both the media, can formally characterise the pattern and help measure the temporal impact of news stories driving the social media reactions.

Case study 3. COVID-19 has dominated social media and news media coverage throughout the year 2020. In comparison, intense public interest in events like George Floyd's death fluctuates, depending upon the latest incidences of black discrimination around the world. The focus of this case study is to track multimodal discourse trends about COVID-19 immediately before, during, and after a defining moment in history, such as the death of George Floyd, which sparked international outrage, given the long history of police brutality towards black people. In this case, George Floyd's death was captured on video, leading to protests worldwide, organised by the Black Lives Matter political and social movement, which advocates non-violent civil disobedience against police brutality and racially motivated violence and discrimination against black people. At this time, the COVID-19 pandemic had heightened social and economic divides, with black and minority groups being much more likely to die from the disease due to poverty, overcrowded housing, and lower-paid and/or key worker roles (e.g., Evans, 2020). The case study examines how these tensions were played out in online UK newspapers and social media.

7.7 Case study 1. Multimodal Approach to Predict Emerging Situations During COVID-19 Pandemic

This section presents a case study using a multimodal framework for news and social media that infers past data to provide early indications of the impacts of pandemic spread in areas such as

new illnesses, recoveries, deaths, and job losses purely based on social media conversations. The proposed Multimodal Analytics Platform (MAP) effectively models the amplification of economic, social, and public health and related emerging issues on Twitter. Analysing social media conversations, MAP discovers language, visuals, and metadata in real-time through complex web searches; then indexes, annotates and aggregates to create interactive reports. New methods are introduced for Information Extraction (IE) and entity disambiguation using multimodal Topic Modeling, clustering, and classification. MAP annotates thousands of images and videos filtered on a time range, personality, headline, or location. Multimodal aggregation of the visual annotations along with the associated textual content and metadata provide an in-depth understanding of the language and visual correlations according to the context (i.e. social actors, processes, and circumstances) and interpersonal stance over time. The proposed method measures growth trends of topics over time by eliminating intrinsic noise in the dataset to accurately predict future eventualities that can contribute to tracking public health and social order surveillance. MAP is hosted on a Google Cloud environment where multiple users can simultaneously work and collaborate using a web interface to provide horizontal scalability of computing prowess as the dataset grows over time. Big Data distributed search along with indexing of each column of a record enables the platform to run data mining and reporting tasks 10x faster than traditional applications not using the distributed environment and indexing of textual data. The case study predicts COVID-19 related metrics based on the observed data in the Twitter dataset. The large-scale *growth and decay* patterns are detected with cues from user-generated Tweets indicative of *new infections*, *recovery*, *deaths*, and *job losses* in the context of COVID-19.

7.7.1 Filtering Data

A method is developed for searching Twitter through webpage crawling that finds matching Tweets related to the list of terms associated with events. Twitter posts first searched with the search term *Coronavirus*, filtering out Tweets based on the search term. Tweets containing a search term *Coronavirus* are further refined based on another set of search terms. Table 7.4 lists search terms on the

topics concerning illnesses, recovery, deaths, and job losses related to COVID-19. However, collected Tweets matching search terms contain a large volume of unrelated records not connected to the topic. Therefore, the prediction dataset discards irrelevant noisy records to improve prediction accuracy. As detailed in the previous section, a series of steps are conducted such as *multimodal Topic clustering, Topic Modelling, n-gram, and word cloud* to provide a deeper understanding of the dataset and eliminate noisy data. Records are clustered using LDA, SVD, or NFM algorithms into a number of topics as a collection of similar Tweets relating to COVID-19 general information, deaths, job losses, etc. To detect the most prominent topics in a dataset, *Named Entity Recognition* and *Parts of Speech Tagging* are used to highlight the prevailing topics using the tags describing events in terms of personalities (who), events (what), places (where), time (when) and the means (how) and establishes connections between the entities. Refer to Figure 7.16.

As presented in the previous section, Tweets are classified into a number of categories (See Table 7.1). Note, records are classified only after data is initially filtered by the search terms. Once a topic is precisely identified by eliminating unrelated Tweets, records are counted for each topic, e.g., new illnesses, deaths, etc. Pairwise comparison is conducted between the number of Tweets on a topic and actual counts for new illnesses, recovery, death, or the number of job losses.

Illness	Recovery	Death	Job loss
ill	recovery	death	job loss
unwell	improvement	passed	pink slip
sick	picking up	dying	axed
not well	betterment	demise	laid off
ailing	come back	loss of life	Fired job
fever	revival	collapse	unemployment
temperature	a turn for the better	passing	
cough	rehabilitation		
smell	return to health		
taste	healing		
not in good shape	rallying		

Table 7.4: Twitter search terms related to illness, recovery, death and job losses.

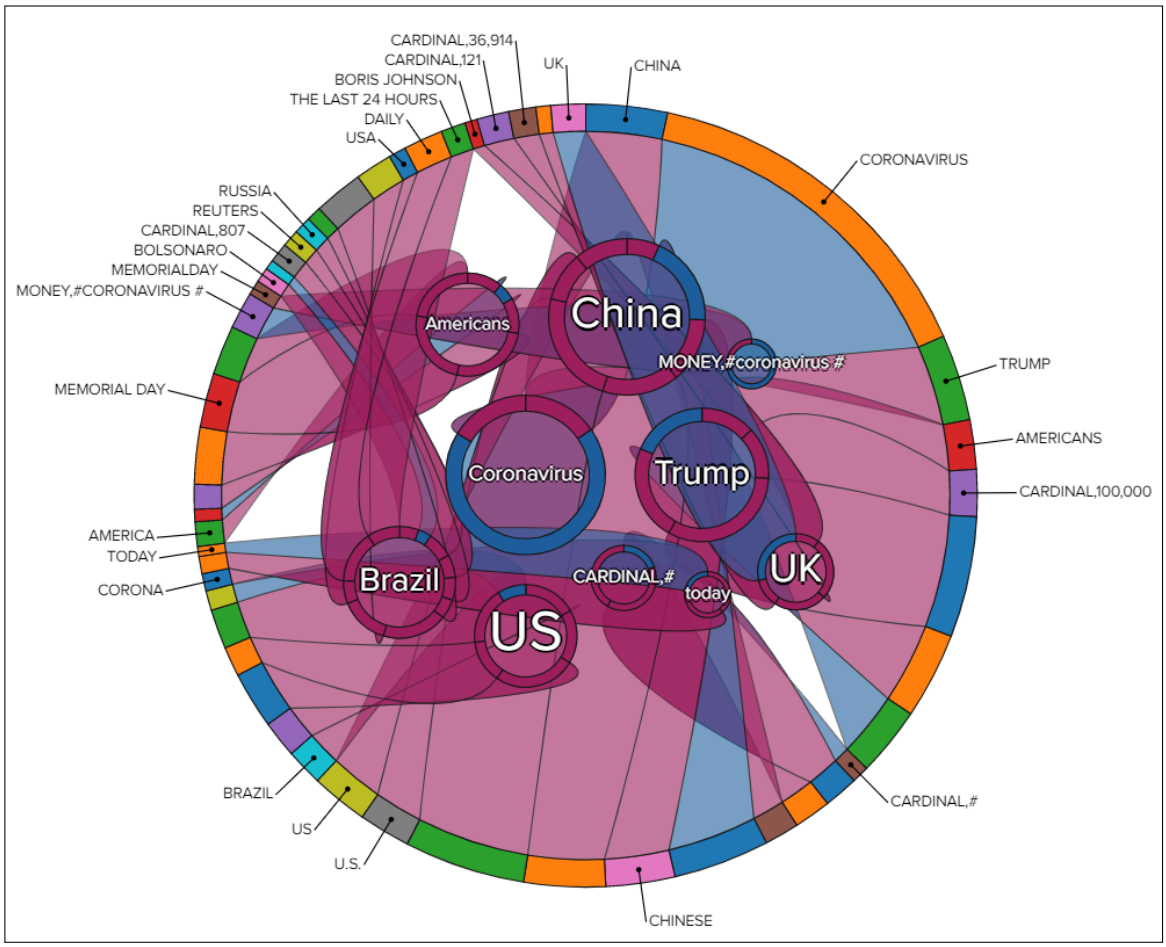


Figure 7.16: **Named Entity recognition** identifies topics in terms of personalities, event, places etc by *Bag of word* representation. The Ribbon chart highlights the most frequent entities and correlations between them which occur within the same record.

7.7.2 Prediction

Growth and decay curves for Tweets volume reveals patterns are similar to the distribution curve of R_0 number for infectious diseases that can either be *linear*, *parabolic* or *exponential*. Pairwise comparison is conducted between social media sentiment and published official statistics about actual counts for new illnesses, recovery, death, or the number of job losses. We can observe that social media conversations effectively capture the underlying latent trends as a precursor to future eventualities. Following the three types of growth/decay patterns, we devised a method for the predictive index using the first and second-order *regression function*. For example, a total number of future infections or deaths (dependent parameter δ_s) can be predicted from the volume of social media posts (independent parameter δ_t) on the closely related topics as given by Equation 7.3.

$$\delta_s = \beta_0 + \beta_1 \times \delta_t + \beta_2 \delta_t^2 + ... + \beta_n \delta_t^n + \varepsilon \tag{7.3}$$

The Twitter dataset used in the experiment in the following *four* time ranges. Each time range consists of one week's Tweets.

1. **Week-1:** 1-8 February 2020, one week following the World Health Organization's (WHO) reporting of COVID-19 as a global emergency on 13 January 2020.

2. **Week-2:** 13-19 March 2020, one-week following WHO's declaration of COVID-19 as a pandemic on 12 March 2020.

3. **Week-3:** 26 May-1 June 2020, the aftermath of the death of George Floyd on 25 May 2020.

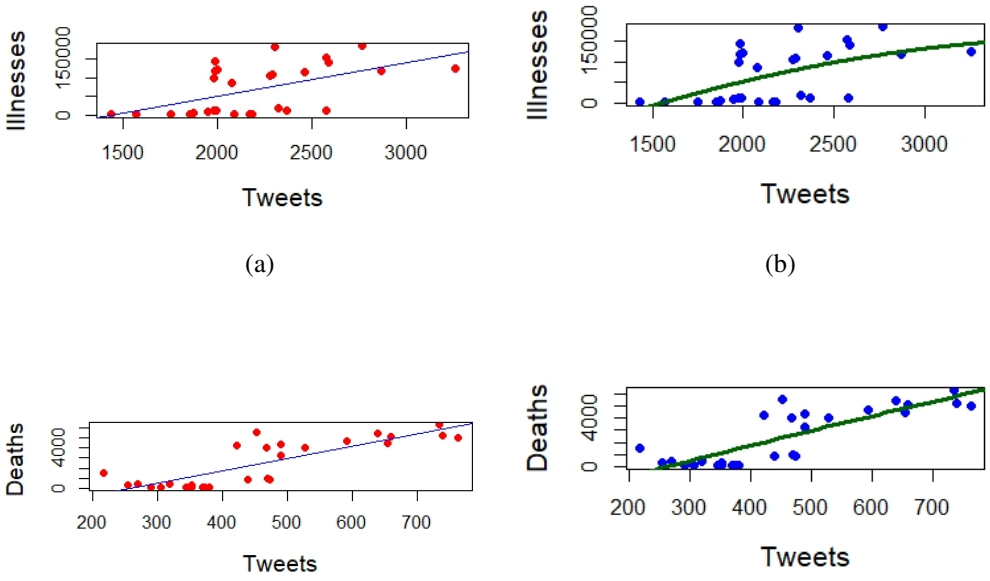
4. **Week-4:** 16-23 June, after the UK opens the shops and restaurants on 15 June 2020, following a lockdown period.

Tweets are filtered based on the keywords related to new illnesses, recovery, deaths, and job losses described in the previous section. Data streams are collected containing geolocation for predicting job losses. Tweets attached with geolocation information originate from the posts where

users willingly share their demographic information. The final dataset for prediction is refined using Search Processing Language (SPL) based on the topic clustering, topic modelling, and classification results. The number of Tweets in each category (illness, recovery, death, and job losses) are correlated with official statistics for predictions. The actual number of eventualities are collected from the published sources, and the US Bureau of Labour Statistics [24,25,29]. To measure relative changes between social media reactions and actual pieces of evidence (henceforth referred to as observed data), the data series (Twitter and observed data) are fit into a linear and quadratic regression model to compute values for *t, p* and R^2. Higher *p*, R^2 values and a lower *t* value reject the *null hypothesis*. The *null hypothesis* is that the two data series (Tweets and observed data) are not statistically relevant for prediction. However, a R^2 *value* and *t value* closer to 1, and 0.05 respectively will establish a strong correlation between independent (Twitter data) and dependent parameter (observed data), and therefore Twitter data can be used for prediction. *Revolution R*, a popular statistical analytics tool, is used to compute values for the parameters.

		Estimate	Error Term	T-Value	P-Value	Multiple R-squared	Adjusted R-squared
Jobs Linear Fit	Intercept	β_0=2.124e+02	1.644e+02	1.292	0.419	0.03806	-0.9239
	t	β_1=1.408e-06	7.080e-06	0.199	0.875		
Death Linear Fit	intercept	β_0=1.408e-06	2.444e+01	13.180	5.10e-13	0.6894	0.6775
	t	β_1=5.606e-02	7.378e-03	7.597	4.59e-08		
Death Quadratic Fit	intercept	β_0=-3.497e+03	2.538e+03	-1.378	0.180	0.6896	0.6648
	t	β_1=1.359e+01	1.091e+01	1.245	0.225		
	t^2	β_2=-1.298e-03	1.087e-02	-0.119	0.906		
Illness Linear Fit	intercept	β_0=1.970e+03	9.369e+01	21.030	<2e-16	0.2976	0.2706
	t	β_1=3.352e-03	1.010e-03	3.319	0.00268		
Illness Quadratic Fit	intercept	β_0=-2.580e+05	2.534e+05	-1.018	0.318	0.3054	0.2499
	t	β_1=2.055e+02	2.216e+02	0.928	0.362		
	t^2	β_2=-2.528e-02	4.762e-02	-0.531	0.600		
Recovery Linear Fit	intercept	β_0=1.193e+01	3.291e+00	3.625	0.0018	0.8034	0.7931
	t	β_1=3.953e-04	4.486e-05	8.813	3.87e-08		
Recovery Quadratic Fit	intercept	β_0=-83332.08	18500.90	-4.504	0.000274	0.8988	0.8875
	t	β_1=6781.82	1165.88	5.817	1.64e-05		
	t^2	β_2=-60.47	14.69	-4.118	0.000646		

Table 7.5: Experimental parameter values for Twitter dataset and observed data relating to new deaths, illnesses, recoveries and job losses due to COVID-19 (linear and quadratic fit). Data series for illnesses and recoveries reject the *null hypothesis* due to high R^2 and low *t* values.



(a) (b)

(c) (d)

Figure 7.17: **Curve fitting illnesses and deaths related Tweets.** Linear (in left) and quadratic fit (in right) as straight and curved lines and observed data as dots for illnesses and deaths for Twitter dataset.

7.7.3 Results and Discussion

The experimental results show connections between Tweets related to illnesses or job losses caused by COVID-19 with the actual official statistics is *tenuous* (due to low R^2 value). On the contrary, reported recovery and death numbers significantly correspond with the related Tweets volume. Therefore, the results indicate that using social media for the prediction can be erroneous for certain topics of interest, such as the total number of new illnesses or job losses. Newly ill people are likely to tend to express their concerns on social media. However, only a tiny percentage of cases are actually diagnosed as COVID-19 in the official records. Therefore, it can be concluded that social media posts can not *always* be effectively filtered to analyse only the relevant dataset. It is also implausible to assume that people who have lost jobs or are likely to do so will publicly post their employment situation. Searching social media posts for meta-unemployment commentaries can be tricky. In that context, we can assume, social media is not a perfect proxy for the issues related to jobs and illness. However, as the experimental results highlight, Twitter data can be used to indicate the number of new deaths or recovery as reported due to COVID-19. During the lockdown, social media is widely used as a platform to broadcast the news to people who are seriously ill. Tweets related to deaths or recovery are posted by those already confirmed with COVID-19, and therefore, data is considered less noisy. Using Equation 7.3 to predict the number of deaths in near term either by linear or quadratic regression functions (Equation 7.4, 7.5):

$$\delta_s = 3.222e+02 + 5.606e-02 \times \delta_t + 7.378e-03 \tag{7.4}$$

$$\delta_s = -3.497e+03 + 1.359e+01 \times \delta_t - 1.298e-03 \times \delta_t^2 + 1.087e-02 \tag{7.5}$$

where δ_s is the number of new deaths or recovery, and δ_t is the number of closely related new Tweets.

7.8 Case study 2. Persistence and Decay of Trends: Dynamics of News and Social Media as COVID-19 Emerged and Spreads.

The case study analyses the trend dynamics of mainstream newspaper articles on the COVID-19 pandemic and their bias in polarising opinions and massive shifts in social media reactions. News media generate a constant stream of real-time content in response to events. However, users' attention is very skewed, and only a handful of news articles become trend-worthy on social media. The underlying scientific principle that drives long or short-term persistence of trends shaping the social agenda is largely unanswered yet. To fill the research gaps, a novel Multimodal Analytics Platform (MAP) is developed that models trends of the news and social media in terms of a *multimodal response function* to current events. MAP provides a joint interpretation of language and visual representations of a big dataset in accordance with linguistic compatibility, semantic similarity, context adherence (social actors, processes, and circumstances), and interpersonal stance over time. MAP can be useful to measure shapes and correlations of the growth functions between news and social media. How to predict social media item growth in relation to the observed news media volume is the focus of this case study. Detailed analysis is carried out on the persistence of long term trend dynamics on COVID-19 in conventional print and social media (Twitter) and also provided a mathematical basis for such trend formulation, surge, persistence, and eventual decline. The analysis of results reports that much of the dissemination of trends originate from established news media, where social media plays the role of a selective amplifier for only a handful of trends.

7.8.1 Growth Dynamics

Growth dynamics of information diffusion in media through the reproduction, amplification, and recontextualisation of the original data follows the same principle as the spread of infectious diseases like COVID-19. The *infodemic* spreads intrinsically through the same model as the *Reproduction Number (R_0)* in pandemics. With the $R_0 > 1$, the disease spreads as one person passes it to more than one person. In terms of information growth, *infection, i.e. diffusion* is the process of

reiterating an existing post by means of share, retweets etc. The popularity of a trending topic is defined as the numerical count of the *number of articles* published on the *news media* relating to the topic *plus* the volume of social media mentions, shares, tweets, retweets, etc. In the case study, we can identify the spread of information through keywords relating to the topic that undergoes a change in volume and semantics over time. Therefore, if N_n is the total number of all the *news articles* associated with a topic and N_s is the amount of *social media coverage*, then growth in popularity at time t is defined as: $N_n + N_s = (N_{n-1} + N_{s-1}) + (\delta_n + \delta_s)$. Where δ_n and δ_s are the incremental growth in number of posts in social and news media since the time *t-1*. The growth of followup articles and posts (δ_n and δ_s) are *growing* and eventually *decaying* over time. It can be observed from the experimental results that the shape of the increment function can either be *Linear, Exponential or Parabolic*, similar to the decaying principle of *radioactivity disintegration* [252]. In the event of *Linear Decay*, news articles are published less often, decreasing at a linear rate until a *cutoff* time is reached. The *cutoff* time is synchronised by either no new development of the prevailing topic, the emergence of a new trend, or the content fails to attract interest in social media. In the event of *Parabolic Decay*, the new publication and popularity shrink at a much higher rate, reducing the volume by almost *four times* in each new iteration than its previous timestamp. *Parabolic Decay*, similar to its shape, declines much faster at the most part after the peak but retains its horizontal, linear shape at the end for a very long period of time. *Exponentially decaying* events are, in reality, another variant of *Parabolic Decay*, where initial loss is more profound than *Linear Decay* but less than *Parabolic Decay* and continues to linger on for very long. However, any decaying pattern, however, *can reincarnate* with a greater surge in trend as a second wave. A generic outline of each of the curve fitting series of data points is provided in Figure 7.18.

The most common way data expands (i.e. the spread of information) is through initial transmission from *news media* to *social media*, followed by subsequent expansion within the social media sphere. However, the other permutations are plausible, such as *news media* to *news media* and *social media* to *social media*. Let us consider the *first scenario*, impact of news media (as independent variable) over social media (as dependent variable). Any increase and decrease in the

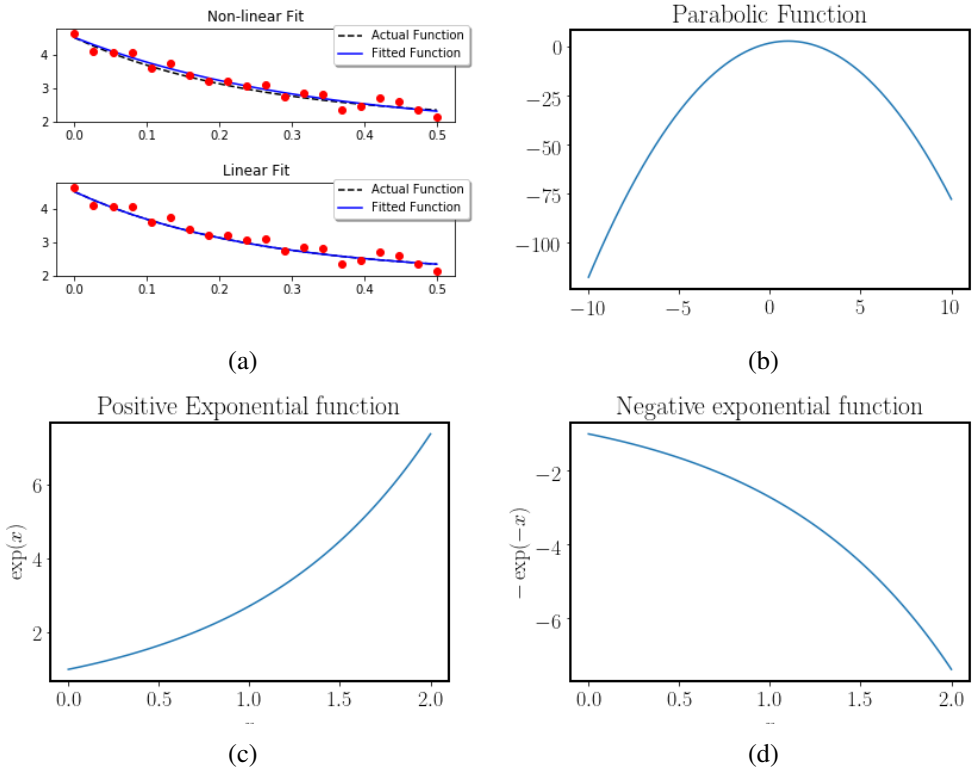


Figure 7.18: Linear and non-linear fit of non-linear (actual) data series (a). Parabolic growth and decay function (b). Exponential growth (c) and decay functions (d).

volume of news articles and the corresponding ripple effect in social media posts are *directly proportional* to each other. If the total volume of decay is x in social and news media, then the linear growth curve equation is given by:

$$f(x) = (\delta_n + \delta_s)p + q \tag{7.6}$$

Where coefficient p is slope of the function and q is a constant. Both δ_n and δ_s are *negative* and the intensity of the slope is determined by the value of p. If the size of news articles and media posts remains constant, then value of δ_n and δ_s are *zero*, therefore, the shape turns to be a horizontal line.

The persistence of trends in social media is measured by the proportion $\theta = \frac{\delta_n}{\delta_s}$. When θ is less than a threshold value k then a topic is assumed to *stopped trending*. The probability of persistence and decay of a social media trend at time t depends on the volume of additional news articles posted on the topic. Therefore, the probability of trending as a linearly decaying function is given by the following equation:

$$\delta_s = \beta_0 + \delta_n \beta_1 + \varepsilon \tag{7.7}$$

For the *exponential* and *parabolic* trend curves, the probability of social media popularity as a dependent parameter to the volume of posts in the news media, is given by the following second-order polynomial function:

$$\delta_s = \beta_0 + \beta_1 \times \delta_n + \beta_2 \delta_n^2 + \varepsilon \tag{7.8}$$

Where β_0 is the *y intercept* coefficient, β_1 and β_2 are the *slope* coefficients and ε is the random error term. The coefficient values are gained with *observed* dependent and independent parameters from the past data (as discussed in Case study 1). Table 7.6 displays the experimental values for the parameters relating to the COVID-19 trend analysis, aggregated on data from Week-1 to Week-4.

Based on the observed parameter values in Table 7.6, probability of persistence (or decay) of social media trend as dependent parameter to that of news media is defined by:

$$\delta_s = 3426.78 + 1548.32 \times \delta_n - 0.000478 \times \delta_n^2 \tag{7.9}$$

Table 7.6: Experimental Parameter Values

Intercept	β_0 = 3426.78	t = -5.1083182	p $<$ .0019
δ_s	β_1 = 1548.32	t = 7.947019	p $<$ .0001
δ_s^2	β_2 = -0.000478	t = -8.841034	p $<$ .0001

7.8.2 Experimental Evaluation

This section evaluates the proposed MAP with the data from *five* leading English UK newspapers (The Guardian, The Independent, The Evening Standard, The Metro, and The Sun) and Twitter. The high-level objectives are to:

- To evaluate the growth dynamics between newspaper articles and follow-up posts on Twitter.
- To predict the persistence and level of contradiction within social media conversations against the impending effects of news media.

7.8.3 Data Preparation

The experiments focus on the long-term persistence of the topics relating to COVID-19 along with the other short-term spikes in trend like the killing of George Floyd and the subsequent *Black Lives Matter* protests. Contrary to traditional tools collecting one-time static data, MAP continually collects news articles (at an interval) through real-time searching on the web. The Twitter database is also searched in real-time for a date range —past or the most recent. Traditional tools using Twitter APIs to query the Twitter database can only return 500 results on each search as the APIs are restricted to a small number of records on each call to Twitter [28]. The alternative method to collect data is by *web scraping* Twitter pages without using the APIs. However, like other popular websites, Twitter use advanced *scrape-detection* software to protect sites from continuously scrapping, since crawling through the pages slows down the websites [239, 298]. These limitations are effectively eliminated in MAP using multiple proxies and integrating the results from several searches. Proxies hide the identity of the actual server while making a data request to the Twitter

server. Using proxies, requests originate from multiple virtual *proxy servers*, without revealing the identity of the actual server that crawls the Twitter pages without violating any terms and conditions of service. The unique data collection technique can reliably collect the maximum amount of data from Twitter without data loss or violating any terms and conditions of service. The results presented here are based on a representative dataset of approximately 1 million Tweets, 50,000 news articles, and 5,000 associated images and videos.

Combining many search results on different topics, date ranges and eliminating noisy unrelated topics is one of the core functionalities of the proposed framework. In this way, MAP measures government responses in different countries towards the COVID-19 using a news and social media data lens. The data is searched and filtered based on the following topics: diagnostic testing, contact tracing, restrictions in the movements, international travel ban, vaccine trials, cancelled public events, school closure, workplace closure, general public awareness campaign (stay at home order), etc. As the media conversation of the events changes over time, the analysis attempts to make sense of the sequence of events and the persistence and the decay of trends.

7.8.4 Results

This section provides fact-checking with the data from five leading UK newspapers and Twitter and characterises the persistence and decay of trends during the time when COVID-19 emerged and spread. The trend is compared for the following *four* time ranges:

1. **Week-1:** 1-8 February 2020, one-week following WHO's reporting of COVID-19 as a global emergency on 13 January 2020.

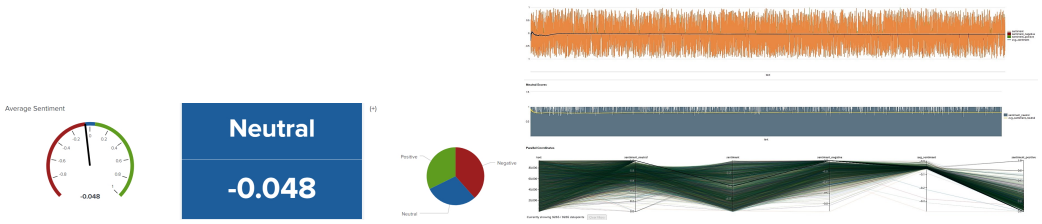
2. **Week-2:** 13-19 March 2020, one-week following WHO's declaration of COVID-19 as a pandemic on 12 March 2020.

3. **Week-3:** 26 May-1 June 2020, the aftermath of the death of George Floyd on 25 May 2020.

4. **Week-4:** 16-23 June, after the UK opens the shops and restaurants on 15 June 2020.

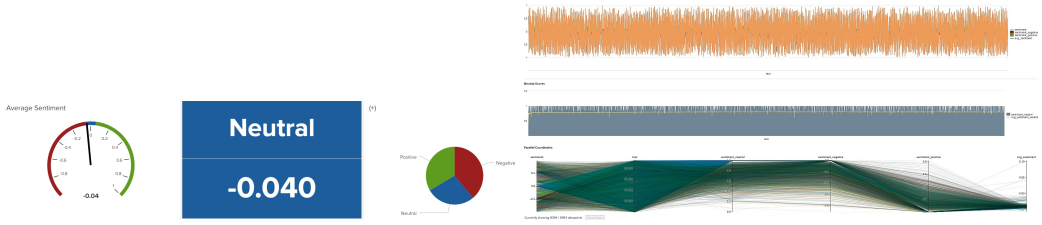
MAP generates dashboards to visualise large datasets with an ability to customise, filter, or combine multiple datasets to create the right kind of reports using *Splunk* and *Python*. The dashboards can be accessed using the following URL: http://35.246.69.64:8000/ (user name: uol, password: liverpool123). The news media responses are based on more than 50,000 news articles published in the five UK newspapers. During the period from February 1^{st} until March 12^{th}, news coverage was only less than 1% of all articles published. In the first full month of January 2020, according to *Time Magazine*, 41,358 news articles worldwide were related to COVID-19 [27]. Out of that, 18,800 headlines mentioned the term Coronavirus. Despite concern raised about the extensive censorship by the Chinese government, information about the virus proliferated much more rapidly compared to the Ebola outbreak one and half years back. In August 2018, when the Ebola outbreak was first reported, only 1,800 English language news articles covered the news worldwide, with only 700 mentioned the term *Ebola* in the headline. This was partly attributed to the fact that the Ebola outbreak was hardly a new trend in 2018, as there has been a previous history of the pandemic lasting between 2014 to 2016. On the contrary, COVID-19 was relatively unknown and a novel phenomenon where everyone was eager for information as the crisis unfolded. Widespread public interest from the emergence of COVID-19 is substantiated by the fact that 1% of published articles generated user views over 13%. However, several prominent analysts and world leaders refuted the trend and criticised the excessive media coverage for COVID-19. For example, Donald Trump criticised CNN news, on a date as late as 27^{th} February 2020, for *"doing everything they can to instil fear in people"*. Similarly, prominent Fox News host Trish Regan accused the media of exaggerating COVID-19. Eventually, *Trish Regan Show* was suspended as the host dismissed concerns about the virus [31]. Despite the initial downplaying of the dangers of the emerging pandemic, the COVID-19 news trend continued to dominate news spaces. Figures 7.19, 7.20, and 7.21 show the *weekly moving average* of Twitter metrics on sentiment, cluster and tag cloud analysis. Results are presented through a joint representation of texts and images by MAP, as discussed before.

The analysis reveals a greater level of engagement with the published news relating to COVID-



(a) (b)

Week-1

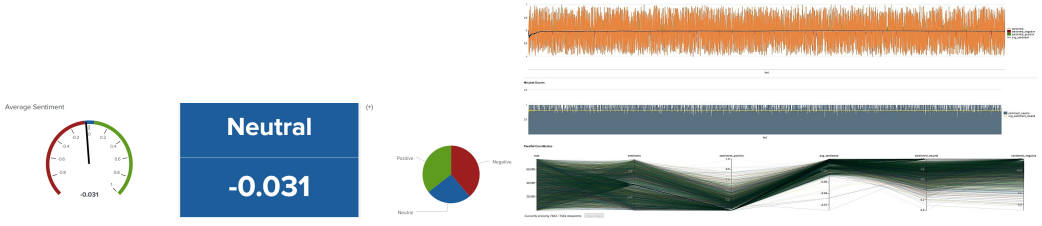


(c) (d)

Week-2

(e) (f)

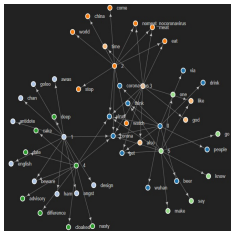
Week-3



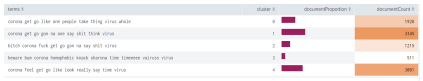
(g) (h)

Week-4

Figure 7.19: Sentiment trend for Twitter data for Week-1 to Week-4. Left side Figures (a), (c), (e) and (g) show an overall sentiment score. Figures on the right side (b), (d), (f) and (h) split the contributing values that make up the score-negative, neutral, and positive. The parallel coordinate visualization (with sentiment scores per text) helps to show how the overall score leans towards positive/negative, given that there are more lines in those areas acting as the tie-breaker.

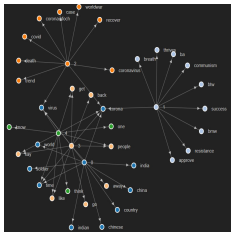


Week-1



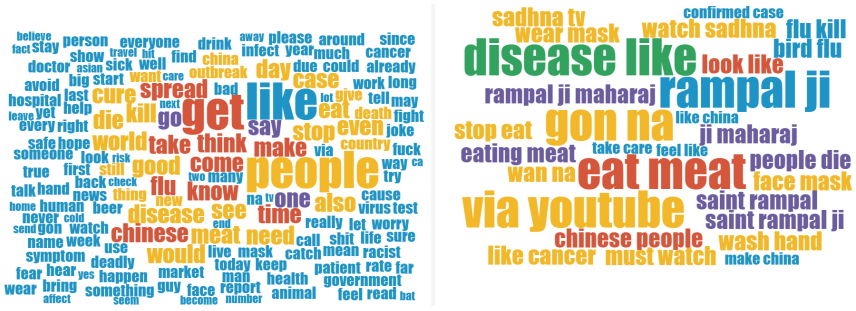
Week-2

Week-3



Week-4

Figure 7.20: Cluster trend for Twitter data by K-Means algorithm for Week-1 to Week-4. Clusters are created based on Co-occurrence of terms. The graph shows the *top terms* for each of the each cluster and how the *top term* counts per cluster compare. Clustering also reveals insights about the terms that exist in multiple clusters and where clusters overlap with each other. Figures on the left are the raw data for the plotting that are shown in the right.



(a)

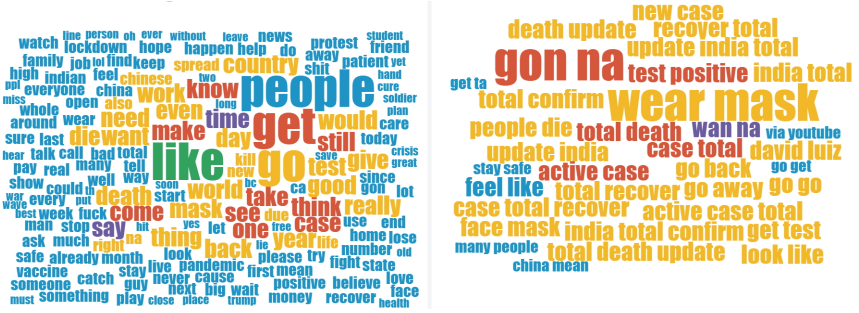
Week-1

(b)

Week-2

(c)

Week-3



(d)

Week-4

Figure 7.21: Tag and n-grams cloud trend for Twitter data for Week-1 to Week-4. Virus related terms such as Corona, COVID etc. are excluded from the results as the common stop words. n-grams which counts the most frequent *n* consecutive words are often more useful than word cloud in terms of detecting patterns.

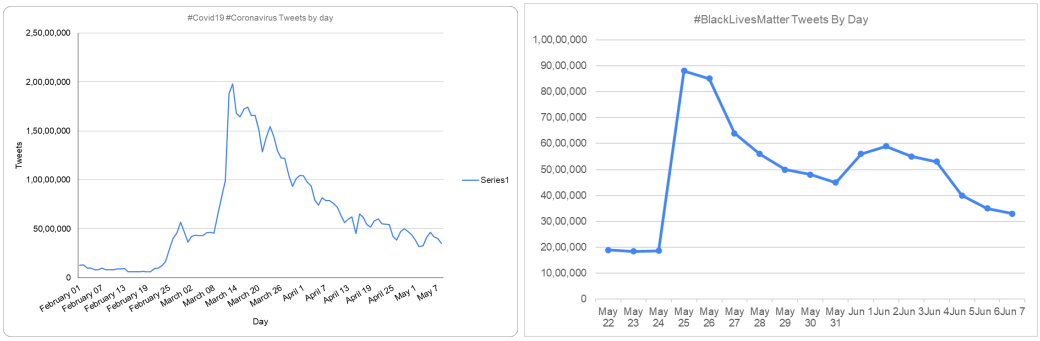
19. As people read more news because of COVID-19, social media trends closely followed the news articles. Following the first reported death by China, WHO's first situation report and the subsequent declaration of global emergency, the first week of February 2020 (Week-1), contrary to the week before that (Week-0), made COVID-19 the dominant topic in social media. The overall sentiment about the COVID-19 remained negative since Week-0 (See Figure 3). However, social media conversation in Week-1, did not *yet* initiate debates about *social distancing* or *self-quarantine*, but instead scrutinized the source and cause of the virus. Subsequently, *No_Meat_No_Coronavirus* turns out to be the most popular Twitter Hashtag relating to COVID-19. Highlights of a few trending conversations were: *Stop eating meat, Coronavirus came from wild or domestic animals*, *China is enforcing the corona quarantine by drone*, and *Corona virus precaution alerts issued by WHO*.

One and half months later, post WHO's declaration of COVID-19 as *pandemic*, during the period of 13-19 March 2020 (Week-2), COVID-19 rapidly spread across the world. Still, the social media opinion seems to have been divided between topics like the restriction of movement and the effectiveness of masks. As expected, panic, anxiety, and loneliness soon settled in the social media sphere. The few most popular Tweets were: *CoVid19 is real. I am on days home isolation*, *This whole working from home is going to change things* and *So many people are going to need help in the coming days*.

The news coverage on George Floyd and the broader issue of *Black Lives Matter* considerably changed the dynamics of the interplay between social and news media. The trend about the George Floyd incident and the ensuing protests completely eclipsed the COVID-19, which has been the massive trend partly due to the fact that it was the *only story*. The week between 26 May to 1 June 2020 (Week-3) saw at least *7 times* more demand in news consumption, and at its peak, social media reactions were *14 times* higher than COVID-19. The reciprocity between and news and social media seems to have altered. Much of the news articles included government statistics and press briefings fed to social media feedback. On the contrary, social media sourced content about grief and protest-induced stories directly from the ground. Unsurprisingly, the peak-trend was *not* short-lived and continued to suppress COVID-19 even after three weeks.

During the *Week-4* between 16-23 June 2020, widespread panic and grief about COVID-19 set the mood of the social media community. A heart-wrenching Tweet accounts for the death and loss: *Just buried my grandma next to my brother. My sister, mama & brother, died in the past three weeks. Found out I got a 14-year-old ill brother. And my two cousins that did not show up to the funeral at home dealing with corona*. Nevertheless, the overall *sentiment polarity* came to be close to neutral, attributing to the government's decisions in the UK and worldwide to open shops and markets.

Figure 7.22 shows the daily distribution of total Tweets relating to *Coronavirus* and *Black Lives Matter* from February to June 2020. The persistence and decay pattern closely resembles the *parabolic* response function with a distinct peak follows by gradual decay and long horizontal ends. Modelling of trend dynamics reflects the uniqueness of each trend. However, a closer look at the growth, persistence, and decay of several trends shows that the life expectancy of a curve can be predicted from the previously observed patterns from the first week of February 2020.



(a) (b)

Figure 7.22: Distribution of Tweets for (a) *Coronavirus* and (b) *Black Lives Matter*. The topic on *Coronavirus* was trending most during the 2^{nd} week of March, while topic related to *Back Lives Matter* reached its peak during the 4^{th} week of May

.

7.8.5 Discussion

The results illustrate that a special *Multimodal Response Function can describe the social media reactions* in relation to published news events regulating the dynamics of social media trends in relation to a trending topic. Much of the trends in social media are the direct consequence of recent news topics as an *external simulator*. However, there are instances where social media *trend* grows all by itself, just like the pandemic that can spread domestically, without an *external carrier*. Readers' level of interest in specific news articles is highly uneven. The majority of news articles receive insignificant amounts of attention from readers, with only very few articles being shared or deemed Retweet worthy over social media. However, right from the start of the COVID-19 outbreak, the dynamics of trend growth in news media profoundly tilted towards the pandemic situation. The news about the virus entirely overshadowed several trend-worthy articles related to Brexit, the political and economic crisis in Venezuela, or refugee problems due to the Syrian civil war.

The analysis reveals differing levels of emotion in relation to major trends in social media. Despite the stress and anxiety caused by the COVID-19, overall *sentiment* detected each week remained neutral (although there is a slight tendency towards negative). On the contrary, the debate around racism and prejudice resulted in the highest level of negative emotions, as observed from the 3^{rd} week's data. Aggregation of sentiment in accordance with the time dimension strongly correlates to the timing before and after the publication of news posts on the same topic in the traditional print media. The results testify that the news events associated with COVID-19 typically lead to an increase in *contradiction level* where opinions greatly swung between positive and negative, bringing the overall sentiment close to neutral. The rationality behind the wide contradiction in sentiment polarity and having the mean sentiment μ_s close to *zero* is the presence of large variance σ_s^2 due to diversity in opinion about different aspects of the pandemic. Using σ_s^2 and μ_s, the level of contradiction is measured by:

$$f(c) = \frac{k \times \sigma_s^2}{k + \mu_s^2} \times \omega(n) \tag{7.10}$$

Where n is the sample size, k $\neq$ 0 is the normalisation constant, and ω is the variable weight function adjusted according to the significance of the comment used for sentiment measure [226, 252, 253].

Social media *catches up* with the *newspaper trends* with some *time lag*. On 30 January 2020, the WHO reported COVID-19 as a global emergency. While the news articles put significant importance on the WHO's announcement, social media fails to react to the news immediately. Experimental results observed the *lag* in overall social media responses during the initial days of the COVID-19 outbreak. Contrary to this, on 12 March, when WHO declared COVD-19 as a pandemic, news and social media coverage almost *immediately* took off exponentially. For the trend on *Black Lives Matter*, persistence and sentiment were *synchronous* in both the media, on the topic of COVID-19. However, newspaper sentiment was significantly more *positive* than social media posts for the entire time duration. As it is apparent from the results, established news media has shown a greater tendency to cover events in a more positive/neutral manner compared to social media, which highlights negativity more frequently.

7.9 Case study 3. Multimodal Approach to Analysing COVID-19 Situations Before and After George Floyd's Death.

The case study analyses data before and COVID-19 and the death of George Floyd and the subsequent Black Lives Matter protests. MAP generates dashboards to visualise large datasets with an ability to customise, filter, or combine multiple datasets to create reports using Splunk and Python. This section provides the results of the text and image analysis for the five UK online newspapers and Twitter for keyword searches for 'Coronavirus', 'Floyd' and "Black Lives' for these periods:
(i) 19-25 May 2020 - One week before George Floyd's death, henceforth 'Week Before'
(ii) 26 May 2020 (British Summer Time) - George Floyd's death, henceforth 'The Day'
(iii) 27 May 2020 - 2 June 2020 - One week after George Floyd's death, henceforth 'Week After'

The news media responses for the three-time periods: Week Before, The Day, and Week After, are based on more than 10,000 news articles published in the five UK newspapers during this period. Besides, Twitter searches for Coronavirus, Floyd, and Black Lives resulted in 208,500 Tweets for these keywords. The distribution of newspaper articles and Tweets for these search terms for the three time periods are displayed in Table 7.7.

Time Period	**Newspaper Articles**			**Number of Tweets**		
	Coronavirus	George Floyd	Black Lives	Coronavirus	George Floyd	Black Lives
Week Before 19-25 May 2020	1,900	0	0	79,575	0	10,063
The Day 26 May 2020	297	2	1	13,665	13,666	6,672
Week After 27 May-2 Jun 2020	1,514	590	42	82,013	72,366	76,135

Table 7.7: The distribution of newspaper articles and Tweets for the three-time periods.

The results for classifications and sentiment analysis, with examples of the cluster and tag cloud analysis for online news and Twitter around the topics of COVID-19, George Floyd, and Black Lives, are discussed below. This first set of results are based on language analysis. Following this, image analysis and multimodal integration between language and visual features are investigated to critically examine the methodology involving linguistic descriptors for visual images.

7.9.1 Classifications

The volume of newspaper articles about COVID-19 in the week preceding George Floyd's death (1,900 articles) decreased during the week following his death (1,514 articles) as attention diverted to this pivotal event (see Table 7.7). Nonetheless, the classification results (see Figure 7.23) showed that the news articles about COVID-19 were primarily concerned with politics and wellness in both weeks. For example, politics and wellness accounting for approximately 30% and 10% of articles respectively in the week before, and this trend continued in the week after. However, the number of articles about COVID-19, which were classified as *black voices* increased slightly (from 4 articles to 13 articles), indicating growing interest between COVID-19 and the Black Lives Matter

movement. We may also see from the 3D cluster diagram (Figure 7.24), that the news articles classified as *politics* and *wellness* are more widely dispersed (and hence disparate) compared to articles that have been classified as *entertainment* which form a distinct cluster.

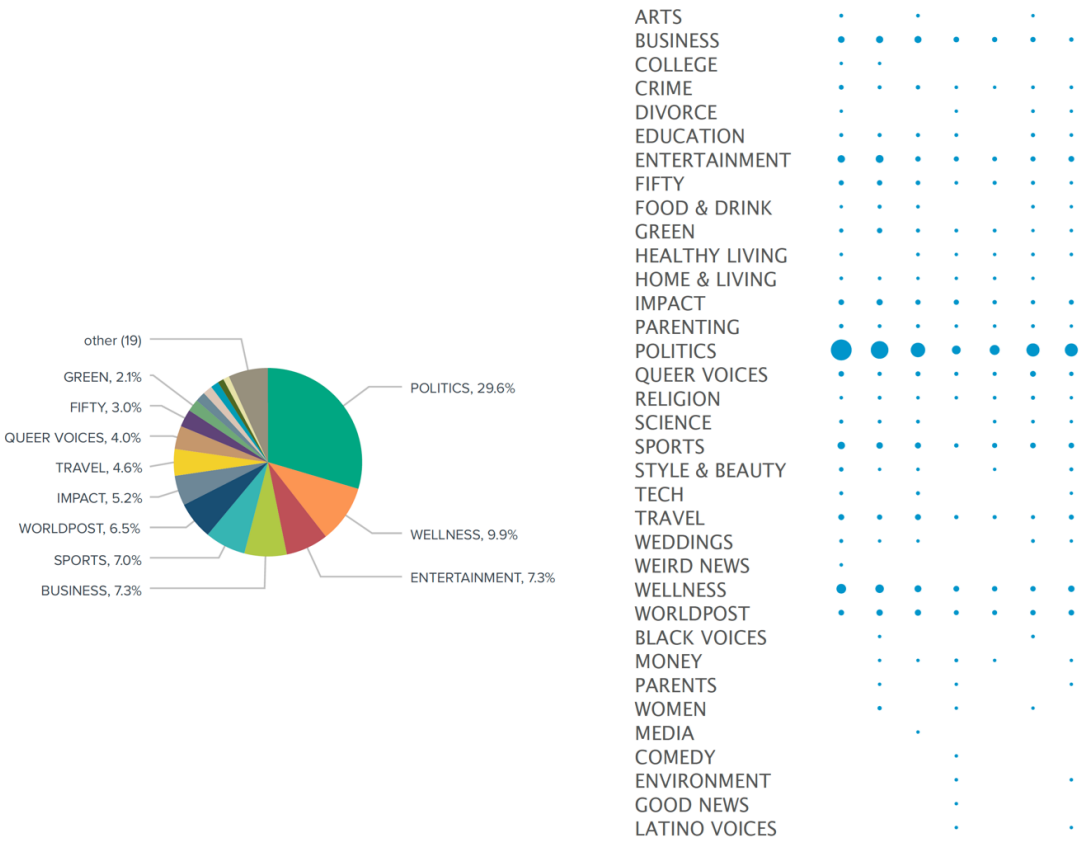


Figure 7.23: Classifications of Newspaper articles with the keyword 'Coronavirus' for Week Before

The results for Coronavirus articles may be contrasted with the articles about George Floyd. That is, in the week following George Floyd's death, the news articles with the keyword *Floyd* (590 articles) were primarily concerned with politics (43.1%), black voices (15.5%) and crime (10.7%), accounting for 70% of all the articles about Floyd. In addition, 42 articles focused on *Black Lives* with classifications politics (45.2%) and black voices (31.0%). In summary, approximately 40% of newspaper articles in the five newspapers focused on Coronavirus, George Floyd, and Black Lives

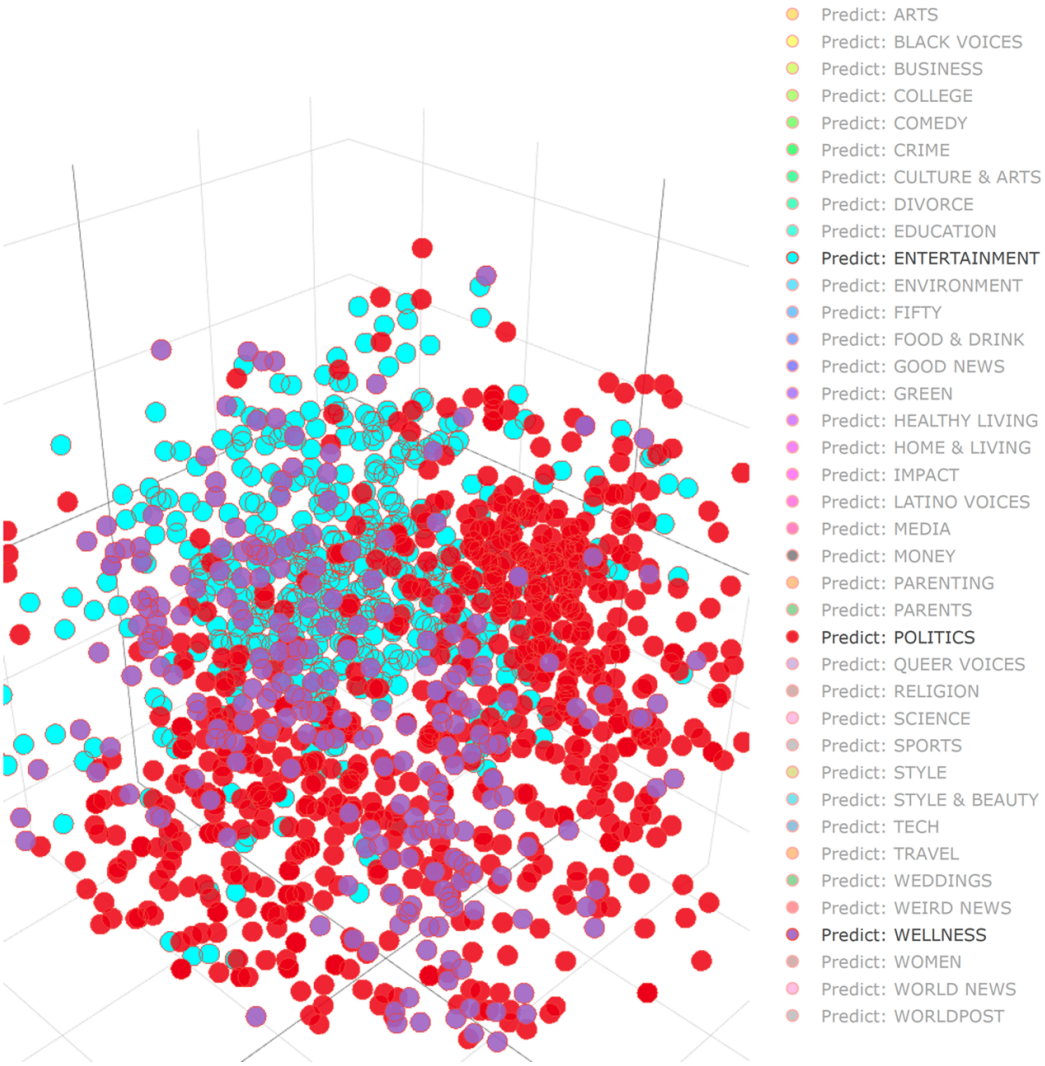


Figure 7.24: Classifications of Newspaper articles with the keyword 'Coronavirus' for Week Before

during this time, with an increasing focus on politics, black voices and crime. The word cloud for newspaper articles for Floyd on The Day is displayed in Figure 7.25a. The reporting is factual and is largely concerned about the circumstances of George Floyd's death. However, the word cloud for newspaper articles about George Floyd in the Week After in Figure 7.25b reveals the shift to reporting about reactions to the event in relation to protests, riots, violence, and the arrest of the police officer.

(a) Newspaper articles on the Day (b) Newspaper articles in the Week After

Figure 7.25: Word clouds for Newspaper Articles on George Floyd

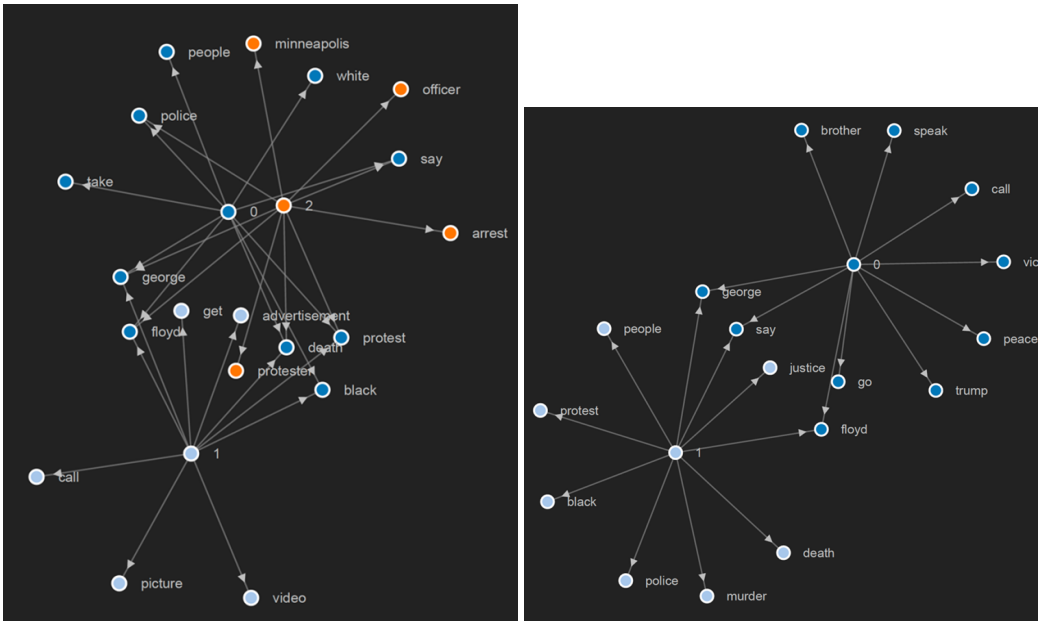
.

The reactions on Twitter to the COVID-19 and Floyd showed differences in mainstream news reporting. For example, Tweets about COVID-19 before George Floyd's death (79,575 Tweets) were primarily concerned with healthy living (28.7%), politics (20.7%), and wellness (14.9%), showing that health was the major concern of the public (as opposed to politics in online news). On the day George Floyd died, Tweets about COVID-19 (13,665 Tweets) showed a similar pattern, focusing on health. Following George Floyd's death, Tweets about COVID-19 did not abate (82,013 Tweets) (unlike news reporting), and the concerns with healthy living (23.7%), politics (14.3%), and wellness (9.6%) remained, but with increased interest in black voices (3.0%). In other words,

the public continued to focus on health-related issues on Twitter whilst turning attention to *Black Lives Matter*.

On the day that George Floyd died, there were 13,666 Tweets with the keyword *Floyd*. The classification results showed that the Tweets were primarily about politics (33.3%), black voices (26.5%), and crime (15.0%). In the week after, the Tweets about George Floyd (72,366 Tweets) moved firmly into the realm of politics (60.4%), black voices (18.7%), and crime (4.6%). The volume of Tweets about George Floyd was less than the number for COVID-19 (72,366 Tweets compared to 82,013 Tweets, respectively). Still, the discrepancy was far less than the difference in newspaper articles about these two topics. In other words, the public reaction to the death of George Floyd nearly matched the level of concern about COVID-19 on social media in the Week After. Subsequent to the defining moment of George Floyds' death, protest marches for Black Lives Matter were organised around the world. The Black Lives Matter movement gained momentum amidst the outrage caused by the killing of George Floyd, supported through social media.

Further differences between newspaper reports and public reaction on Twitter are evident in K-means clustering results, displayed in Figure 7.26. Clustering uses a Topic Modelling algorithm that identifies and groups homogeneous terms based on the topic similarities in semantics. Clustering results show that the newspapers articles concerned with Floyd during the week after show interacting clusters of terms, with the common terms "George", "Floyd", "death", and "protest" in three clusters being linked to the terms "black" and "police" in two clusters, as displayed in Figure 7.26a. On the other hand, the Twitter clusters are distinct, with terms "George", "Floyd" and "say" interacting with the terms "brother", "call", "go", "peace", "speak", "trump" and "violence" in one cluster and "black", "death", "justice", "murder", "people", "police" and "protest" in another cluster. In this way, we can see that members of the public specifically link wider political issues involving US President Donald Trump and murder to the killing of George Floyd on Twitter.



(a) Newspapers clusters for Floyd. (b) Twitter clusters for Floyd.

Figure 7.26: Cluster Analysis for newspapers and Twitter in the week after George Floyd's death.

7.9.2 Sentiment Analysis

The overall sentiment in newspaper articles about COVID-19 tends towards neutral and low negative values, as positive and negative sentiment provide counterbalances to each other, given the different aspects being reported (e.g., death counts, government initiatives, and good feel news stories). Also, at the beginning of the outbreak, due to a lack of scientific information, public opinion was broadly divided about the long-term impact of COVID-19. Many prominent figures, like Donald Trump and news media outlets such as Fox News, downplayed the pandemic situation. Therefore, a large variance in sentiment polarity with a higher level of contradiction resulted in the overall sentiment values being close to neutral. A similar trend is found in Twitter posts about COVID-19, where members of the public debate different aspects of the pandemic with divided opinions. However, the same cannot be said about the online reporting and Tweets about George Floyd. As displayed in Figure 7.27, the sentiment score for online newspaper articles about Floyd

is -0.513, and this increased to -0.643 in articles during the week following his death. The sentiment coordinates in Figure 7.28 show the distribution of the weightings across the different articles, with the average sentiment values exceeding -0.5 in terms of value.

Figure 7.27: Sentiment value for newspaper articles about George Floyd on 26 May 2020.

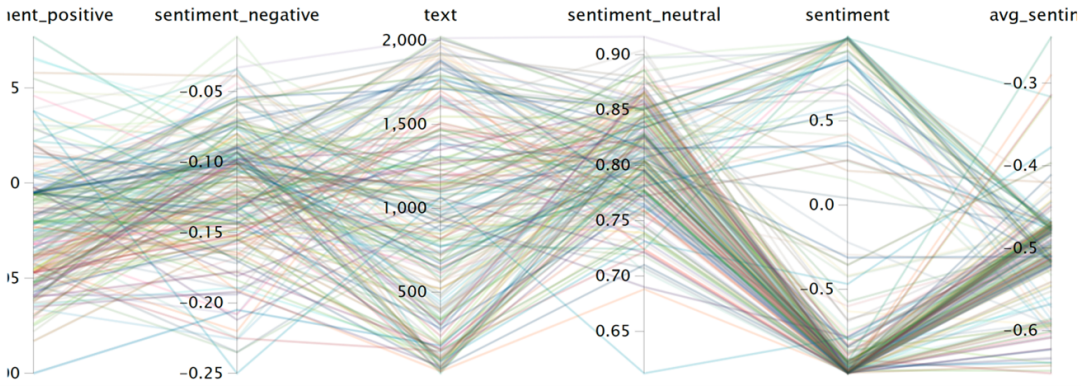


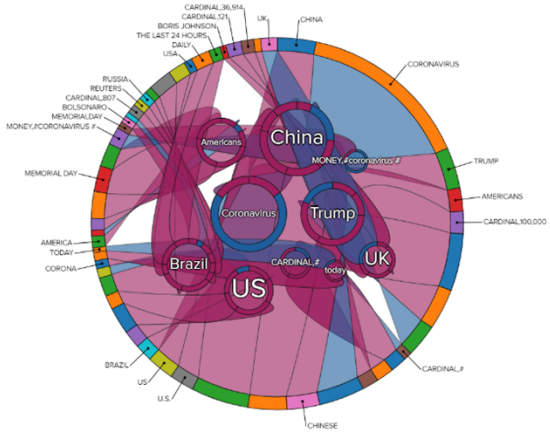
Figure 7.28: Sentiment coordinates (positive, negative, neutral, and overall score) for newspaper articles about George Floyd on 26 May 2020.

The sentiment analysis of Tweets about George Floyd has a negative score of -0.381 on the day and -0.158 during the week, which follows. These somewhat surprising results perhaps reflect the nature of the postings on Twitter, including calls to rally, rather than the actual sentiment expressed in the Tweets. The sentiment for Black Lives has similar values, with low levels of

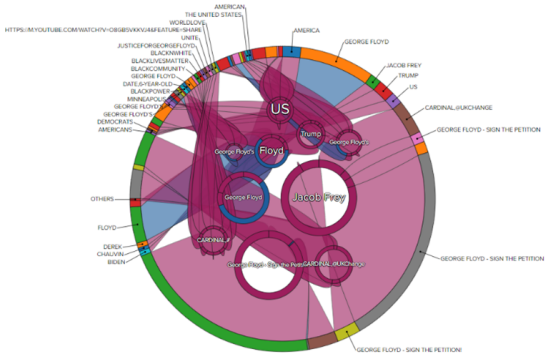
negative sentiment in the Week Before and Week after (e.g., -0.091 and -0.060, respectively). However, the outcry of negative sentiment rose significantly on The Day when George Floyd died (-0.268), which galvanised members of the public to take action, which resulted in protests around the world.

7.9.3 Identifying the key topics

Information extraction is a method that accepts language as input and provides predefined sets of elements such as places, people, organisation after entity disambiguation. Compared to news articles that are professionally written and diligently edited, Tweet posts are short and, in some cases, ambiguous without knowledge of the context. In language analysis, making sense of Tweets involves a processing pipeline of language recognition, tokenisation, Part of speech, and Named Entity tagging to improve *informativeness* by extracting domain knowledge in terms of who, what, when, where, how, etc., in order to understand key information. Conventional analytics systems break the records into words and place them together either by putting all entities into a single bucket (one-hot vector) or separate buckets for each record (bag of words) [124]. A collection of words, without relation to other words, do not retain their semantic meaning. Therefore, MAP adopts a different approach by identifying up to 19 Named Entity Recognition (NER) and 37 Parts of Speech (POS) tags while maintaining explicit relationships between the tags. The accuracy of the POS and NER tags are manually verified against the ground truth. As shown in the ribbon chart (Figure 7.29), NER tags and the correlation within them, extracted from Tweets, before and after George Floyd incidence, noticeably captures the shift in focus as the conversation tilted towards the USA, Black Lives and president Trump from COVID issues in China, Brazil, USA, and the UK. However, the analysis of the image tags, which aggregates the generic descriptions of the thousands of images, failed to reveal such shifts in focus (see Figure 7.29, 7.30), again pointing to the need for more sophisticated image processing techniques.

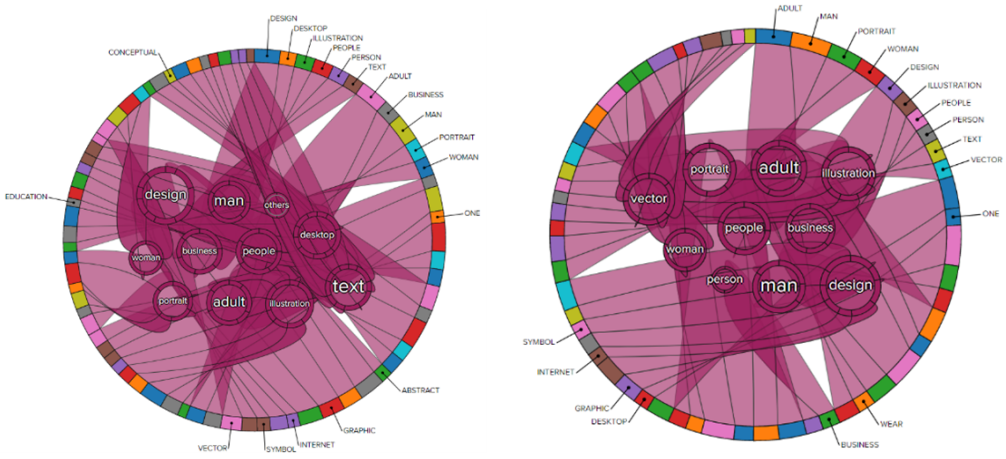


(a)



(b)

Figure 7.29: Named Entity Recognition for Twitter texts before and after George Floyd's death.



(a) (b)

Figure 7.30: Named Entity Recognition for Twitter images before and after George Floyd's death.

7.9.4 Discussion

As evidenced by this study, computational approaches to the multimodal analysis of large datasets is a complex undertaking involving multiple processes, including data collection, data storage, data cleansing, data analysis of language and image, multimodal integration of the results, and subsequent classifications and visualisation of the results to understand patterns and trends. In this case study, online newspaper articles and Twitter data were collected. Natural Language Processing tools and image processing tools were applied and integrated the results to interpret differing reactions to COVID-19, George Floyd, and Black Lives Matter. In doing so, the case study revealed that despite the stress, anxiety, and damage caused by the COVID-19, the debate around George Floyd's death and racism and prejudice resulted in the highest collective level of adverse reactions in online news media and social media, compared to the polarising and highly differentiated response to the pandemic, at least in the period under consideration (i.e. 19 May to 2 June 2020). Besides, the results revealed the primary concerns about COVID-19 related to health and well-being in Twitter, compared to politics in the online newspapers.

However, the key issue arising from this case study involves developing a theoretically-informed methodology for visual analysis, which can be integrated with linguistic analysis. Current image

recognition algorithms depict images in terms of abstract captions, which involves a significant reduction in meaning which semiotic analysis can capture. Moreover, in the case study, the *Clarifai* image captions do not identify the celebrity names (although these models exist) and the place of the event. For example, the image tags for Donald Trump are: administration, people, business, leader, politician, man, democracy leadership, parliament. These tags are then categorised into a single classification label as 'politics'. While this result may be useful in some cases, further investigation is needed to expand our approach to incorporate more sophisticated image processing algorithms to identify personalities, places, and events more precisely in order to establish relations between these key participants, processes, and circumstances.

Further to this, the current scope of image recognition needs to be extended beyond classification to analyse sentiment, topic modelling, and other dimensions that are captured through linguistic analysis. Despite these issues, this study has involved developing an online platform with facilities for integrating tools and methodologies discussed throughout the thesis for multimodal approaches to big data analytics to reveal what can be achieved computationally at this point in time. The challenge remains to develop new tools and approaches to addressing the evident shortcomings of what can be achieved at present.

7.10 Summary

This concluding chapter of the thesis offers *three* more additional input data modalities: language, visuals (image and video) and textual metadata along with the existing stream and batch integration as demonstrated in preceding chapters for comprehensive coverage of a dataset and provides a more realistic real-world representation. This chapter introduced novel methods for information extraction and entity disambiguation using multimodal Topic Modeling, Clustering, and classification. A new multimodal framework was proposed for a big data predictive system that correlated cues from *three modalities* to provide early indications of eventualities. Multimodal entity recognition and tagging were introduced to eliminate noisiness in data. Multimodal classification, entity disambiguation, and clustering helped formally characterise the cohesion, discord, or reciprocity

within different sets of arguments in the dataset. A novel Multimodal Analysis Platform was developed using the proposed methods to derive critical attributes from multiple modalities. Advanced levels of *information feature fusion* enabled by the latest technologies from Data Science and Cloud Computing empowered the platform towards a next-generation analytical platform that is both powerful in functionalities and easy-to-use.

The proposed platform exploited the cross-modal integration of disparate datasets across social and news media. The platform provided facilities for multimodal integration of disparate datasets across social and news media, as demonstrated through the analysis of articles in five UK newspapers and Twitter data about COVID-19, George Floyd's death, and the ensuing Black Lives Matter protests. Sentiment analysis revealed higher than usual levels of contradiction in general public opinion about the COVID-19 pandemic compared to the negative reaction to George Floyd's death. Empirical observations indicated how multimodal analysis of news reports and social media data could reconstruct the impact of the pandemic and acts of violence on society.

Using the proposed multimodal integration techniques such as multimodal topic modelling, clustering and classification, a case study was presented on the nowcasting (i.e. near term forecasting) using digital traces for signs of new illnesses, recovery, job losses, and deaths purely backed by the ebb and flow of social media conversations. *Four* week's Twitter dataset is analysed, aggregating one week's Tweets after *four* important milestones as the COVID-19 situation unfolded. Tweets were filtered based on the keywords relating to new illnesses, recovery, deaths, and job losses. The dataset was further refined to improve prediction accuracy using Topic clustering, Topic Modelling, n-gram, and word clouds, which provided deeper reasoning of the dataset to help isolate the relevant dataset from the unrelated noisy records. Once a topic is accurately identified by eliminating unrelated Tweets, the number of Tweets were counted for a topic such as new illnesses, recoveries, job losses, or deaths, and a pairwise comparison is conducted between the number of Tweets in a topic and actual counts of eventualities. We can observe that social media conversations effectively captures the underlying latent trends as a precursor to future eventualities. Following the three types of growth or decay patterns, a method was devised for predictive

index using the first and second-order regression function. Based on empirical observations, we can theorise that a reliable prediction system for COVID-19 can be developed based on the dataset consisting of social media conversations. Thus, we can conclude, a multimodal response function can effectively describe social media reactions in relation to current events.

Another case study on COVID-19 showed the trend dynamics of the social media response as a consequence of published news events. Persistence and decay of trends can be mathematically modelled by linear and polynomial regression functions (exponential and parabolic) or any possible combinations of them. Experimental results established evidence for the growth model on COVID-19 and Black Lives Matter. Despite the unusually long term persistence of trends, the stochastic process, in reality, exhibited finite variance in the expected life expectancy of the curve. Therefore, despite uniqueness in every pattern, we can speculate the *decay* of a trend can be predicted if the *growth* trend resembles any of the previously observed patterns.

Despite the new innovations in multimodal discourse analysis introduced in the chapter, the actual computational overheads of machine learning and NLP models are found to be scaling up more swiftly than the (known) lower bounds of theory, suggesting that further improvements towards faster computation might be possible as a future research direction. This will remain a critical research agenda for the foreseeable future.

Chapter 8

Conclusions and Future Work

This Chapter reviews the main contributions of this thesis in Section 8.1, followed by a discussion of a number of possible future work directions in Section 8.2.

8.1 Summary of Contributions

The primary aim of this thesis was to attempt to answer *the critical research question* defined in Section 1.2:

How to provide a joint interpretation of several knowledge representations in terms of multiple modalities such as stream, batch, linguistics, visuals, metadata creating a unified view that produces a more accurate and comprehensive representation of data compared to a single standalone knowledge base?

The overall contribution of the thesis is the Multimodal Analytics Framework (MAF) constituting of *there* major components: data ingestion, Multi-criteria Decision Making (MCDM) and Multi-agent Lambda Architecture (MALA). The MAF is subsequently implemented as a web-based platform in the cloud environment called Multimodal Analytics Platform (MAP). In summary, the proposed MAF set out to provide an end-to-end solution for the state of the art big data analytical challenge in *five* chapters as follows:

Chapter 3 introduced novel big data ingestion techniques. *The key innovations* in this chapter are a fault-tolerant, horizontally scalable big data ingestion framework hosted on the cloud environment and mining and searching patterns in it while data is *in transit*. At batch-stream setting, a variation of the Expectation-Maximisation algorithm with the Gaussian Mixture Model was proposed to soft allocate every streaming window data to a new or existing cluster distribution. The chapter also presented predictive modelling through a *Higher Order Markov Chain* and a live streaming data storage mechanism to support a low-latency, highly available stream processing architecture.

Chapter 4 presented a unique Multi-Criteria decision-making framework through a graph-based approach that finds the best-fit tools and methods on the ingested data described in the previous chapter. A novel fuzzy graph-based framework was proposed to model a set of criteria and problem solvers with real-world unpredictability. The model produced benchmarking results for each problem-solver in terms of absolute values to support decision making. The model was designed to solve the *variety* problems due to multiple input data formats and schemas in big data through a dynamic selection of methods most suitable for the current window of stream data. Experimental results showed that the proposed model produces better insights compared to a single static method by employing different methods for each discrete set of data. Nevertheless, the model also impacted negatively due to higher response time, space, and cost overheads. This induced a 5% delay in the response time on average and required additional storage space to operate, and incurred an overall higher maintenance cost as all the alternative components required running simultaneously.

Chapter 5 introduced a novel Lifelong Learning model through Multi-agent Lambda Architecture (MALA), a collaborative ensemble framework for stream and batch data. The proposed MALA architecture retains the past learned knowledge and transfers it to future learning, transforming the process into a Lifelong Learning System. The model can update the dimension-reduced feature set in every streaming sliding window without requiring to rebuild the entire training set from scratch. In addition to incremental learning methods, this chapter provided a semi-

unsupervised *lifelong sentiment classification* method. The method's performance often exceeds supervised learning, which shows that the knowledge reusing of lifelong learning is beneficial.

Chapter 6 explored a new collaborative ensemble framework to improve Lifelong Machine Learning results. Efficient collaboration between batch and real-time components deliver deeper insights at low latency. The model gets precise over time with an incremental learning approach. The initial base learners adapt faster in smaller intervals of a sliding window, improving the accuracy rate by countering the *concept drift*. A weak learner was boosted by a weighted sum of learners from the previous iterations. A *weighted hybridization* strategy with multiple ensemble parameters was validated by implementing a *Recommender System*.

Chapter 7 presented a new method for joint interpretation of multiple knowledge representation in terms of linguistics, visuals, and metadata and created a unified view to produces a more accurate and comprehensive interpretation of data compared to a single standalone knowledge base. The chapter developed deep semantic interpretation algorithms for cognitive computational vision based on the new innovations for image, text, video, their relations and context analysis. The chapter introduced new methods for multimodal classification, Topic Modeling, entity correlation, and clustering to train and predict using multiple modalities. The chapter also develops an integrated semi-automated open and extensible prototype of a multimodal system in a cloud environment. The proposed methods were validated using *three* case studies on a COVID-19 dataset.

From the perspective of the key research question, the thesis reveals the necessity of using an end-to-end multimodal theoretical framework to integrate the various computational tools into a robust methodological approach for harvesting information.

8.2 Future Directions

Based on the foundations of the hybrid model between the stream and batch processing, lifelong learning architecture and multimodal correspondence between text, image, video, and metadata laid out in this thesis leads to extend the current scope of multimodal analytics research. The future research area on *Multimodal Rhetoric in Online Media Communication* could leverage the current

work of the relationship between different modalities as a joint representation of text, image, and video. The future research will deliver a Proof of Concept methodology and functional computational system building on multimodal discourse analysis, sociopolitical models of rhetorical effects, and computational deep semantic processing of language, images, and their combinations. On the one hand, these together could augment results being obtained from computer vision and natural language understanding systems, content-based video retrieval, and machine learning and, on the other hand, provide detailed discourse-based tracking of messages and their reinterpretations. For this, the thesis offers a highly developed and finely differentiating account of how meaning arises from integrating language, images, and other resources in texts, interactions, and events. The combined approach thus aims to resolve the gap between highly-detailed, contextualised analyses of small samples of multimodal texts on the one hand, with highly-aggregated, decontextualised big data approaches (e.g. reductive content analysis) on the other, by leveraging the Multimodal Analytics Platform (MAP) developed as part of the thesis and emerging multidisciplinary theories and techniques of multimodal analysis.

Based on the proposed MAF in the thesis, building a novel broad cross-disciplinary consensus concerning the research questions and methods for the fine-grained tracking of mutually constitutive relations between data analytics techniques and their multimodal medial construction is now an urgent research task. Future research could combine proposed conceptual frameworks critical for achieving advances in this area at this time, aided by proposed computational techniques and making those techniques accessible to a far broader range of involved actors. It is observed that this kind of trans-disciplinary collaborative activity is a primary goal and primary deliverable of future research.

8.2.1 Context for Future Investigations

As laid out in the thesis, in multimodal discourse analysis, the study of the meaning arising from the integration of language, images, and other resources in texts, interactions, and events has emerged as an interdisciplinary field of research, providing powerful analytic frameworks for analysing

how language and images combine to communicate meaning. However, large scale analysis and corpus-based empirical grounding and testing of insights are required to map discourse patterns and trends.

Future research could combine existing research laid out in the thesis in order to provide powerful new triangulation techniques building on a multimodal discourse analysis of language and images. Existing methods in the thesis will be further enhanced with theories and concepts from visual communication and critical discourse analysis. Given the availability of new technological opportunities in the thesis, new methods for combining the contributions of textual, visual, and aural contributions in relation to context are now essential.

Based on the advancements in the thesis in terms of multimodal integration, future researches could incorporate video processing algorithms with image and language processing tools, semantic classifications, and machine learning techniques into an open and extensible prototype multimodal system drawing on several existing prototypes and system components from the participating partners. The methodology that is already developed as part of the current thesis could further expand in the future in terms of the proposed multimodal classifier, machine learning and neural network, which could be used to classify the text, image, and video relations in terms of co-contextualising relations (similar meanings) and re-contextualising relations (different meanings) in relation to context.

Given the progress made with the thesis, in future, it is a crucial research task to address key issues involving multimodality in the digital age, as proposed here.

8.2.2 Further Work

One potential line of research could be to investigate how language, images, and videos are used to developing and delivering a prototype multimodal system that uses image, language, and video processing tools, semantic classifications, and machine learning. Effective ways of using and improving the current system by applying theories and methodologies from cultural sociology, visual communication theory, critical discourse analysis, and computational methods could be explicitly

targeted in the future line of research.

In summary, the specific aims of the future research are:

1. To develop a novel integrated semi-automated open and extensible prototype multimodal system for analysing images, text, videos and image/text/video relations according to context to reveal the strategies through which rationalities are variously constructed to maximise popular appeal across different political spectrums in online news media reports.

2. Proof of Concept for theoretically informed empirical techniques for the semi-automated contextual analysis of text, image, and video relations in online communications.

The combined expertise of the models developed through existing research covers the levels of abstraction and analysis necessary for addressing the future research questions identified.

Given the progress made with the thesis, in future, it is a crucial research task to address key issues involving human activity in the digital age, as proposed here.

Appendix A

Deploying the Model into Cloud

A.1 Introduction

The models developed in the thesis are deployed and hosted into a cloud enrolment instead of a local server. Hosting the applications on the cloud has brought a number of advantages to this research. Most importantly, models are developed focusing on the application logic rather than managing infrastructure. Big data cluster management bears significant overhead in terms of administration, failover, and deployment. In this chapter, an *Automated Cloud Deployment* (ACD) model is proposed, which automates cloud deployment from a graphical user interface. Multiple users can access the project, and a *Role Based Access Control* mechanism is proposed for different user groups. The proposed methods are a *one-time* development effort, which saves time on each occasion the source code is modified and required to redeploy the application into the cloud. ACD is a web-based application that allows for the provision, management, and service for the big data analytics stack in a cloud environment. ACD is also *cloud-agnostic* and works well with *AWS, Google* and *Microsoft Azure* cloud environment.

A.2 Objective and Requirements

The objective of the ACD is to automate the repetitive task of infrastructure setup into a cloud environment. For example, a *Hadoop cluster* creation can be initiated from a web UI by selecting the required configuration for the cluster in an automated way. ACD is configurable through a graphical user interface. A set of users are created by an admin, who also links the logged-in user to a group, i.e. *Admin, Manager, Normal, or a Support user*. An admin can create a user profile that consists of a password to access the cloud environment *pragmatically* along with attributes attached to each user (see details below). The logged-in user can deploy multiple projects into the cloud. The logged-in user starts with creating a project which requires the *IP addresses* in the cloud for deployment and network related information such as CIDR or DNS [42]. The logged-in user then *selects* a *SKU* from the list of available option. For instance, the logged-in user can select *Hadoop cluster creation* as an SKU. Once the project and SKUs are created, the deployment process can be started from a user interface with a click of a button. The user interface has a *deployments view* which displays all deployed projects on the cloud and monitors the various resources and utilization of the cloud space. In addition to SKUs, the logged-in user can process any *service* like executing a script, component or cookbook that may additionally be required for the SKU. In addition, the user can perform *administrative tasks* using *management console*, can debug through *logs* and generate *reports*. Details of each task are given below.

1. An admin has the privilege to log in to Active Directory (AD) and view, manage and modify the following:

   (a) Users,

   (b) Projects (Available projects including deployed and non-deployed),

   (c) SKUs,

   (d) Deployments (to view those projects that are deployed),

   (e) Services,

(f) Management services interface enables viewing, modifying, and managing components such as service category, AMI IDs, scripts, and AWS resource templates. Using the management services, a user can view components such as subnets, security groups and environment types,

(g) Logs,

(h) Reports.

2. Easily provision, manage, and service through an intuitive user interface,

3. Provide service offering in a *cloud agnostic* way,

4. Provide multi-tenant (isolation) [196] features for the user to self-manage deployment.

A.3 User Management

In the *user interface*, an admin can view an existing user, create new users and modify/delete a user.

1. The logged in users can create a new user. The following are the required fields to create a user:

   (a) First name,

   (b) Last name,

   (c) Access key,

   (d) Secret key,

   (e) Phone,

   (f) Email,

   (g) Address,

   (h) City/Zip.

2. The logged-in user can view a list of projects created by clicking on the *View Projects* link.

A.4 Projects Management

Multiple SKUs are combined to create a project. In the *project management* interface, the logged-in user can view, create a new project, modify/delete projects and deploy projects. The logged-in user can preview and modify the SKU components while creating a project. The logged-in user can add additional services to the already deployed projects.

3. Creating a project requires the following fields:

   (a) A user from the dropdown.

   (b) A project start and end dates.

   (c) A project short name, which is restricted to 4 character length and unique for a user.

   (d) Full names in a text box (limited to 24 chars).

   (e) An environment type from a dropdown list.

   (f) A cloud provider from a dropdown list.

   (g) A default and backup regions from a dropdown list.

   (h) A CIDR value from a popup or dropdown list.

   (i) A DNS support value from the dropdown list.

   (j) Enable hostnames value from the dropdown list.

   (k) An Instance tenancy value from a dropdown list.

   (l) A SKU from a dropdown list.

   (m) The logged in user can preview the selected SKU and change attributes of services such as node count, instance types.

4. The logged in user can submit projects for approval. Approvals can only be granted by the logged-in user who has a *Manager* or *Admin* role. (when a project is saved, a complete copy of the SKU is saved along with the project).

5. The logged in user can provide additional services to the already deployed project and redeploy it.

6. The logged in user can filter the active projects or choose to view all the projects, including those are *logically deleted projects*.

7. The logged in user can filter the projects based on the user id by typing the id in a search box.

8. The logged-in user can view the user details by selecting a user id.

9. The logged-in user can view the current status of the project.

10. The logged in user can delete a project. Deleting a Project deletes the stacks associated with it, in the case of AWS. The logged-in user can view deleted projects. A deleted project cannot be redeployed again.

11. A new key pair is generated for each unique project.

A.5 Selecting an SKU

Multiple services are combined to create an SKU. On the SKUs page, the logged-in user can view, modify/delete the existing SKUs. User can create a new SKU by adding a set of services having cookbooks and scripts to run. The following actions are required to create an SKU:

1. Clone the available SKU as a template to create a new SKU or choose a default template.

2. Type-in a unique name for an SKU. The system checks if the provided name already exists and alert the logged-in user.

3. A predefined set of categories (tabs) are shown to the logged in users like— *Analytics, Ingestion, Data Lake, Data Store, Enablement, and Monitoring*.

4. Select a category and add services in that category to the SKU.
5. Modify the order in which the services are run
6. Change attributes of a service such as a node count, instance type.
7. View a list of SKUs with pagination.
8. Search an SKU by typing the SKU name in the search box.
9. Modify an SKU and the services attribute to run.
10. Delete an SKU, and the system marks the SKU as deleted.

A.6 Deployments Management

Admins can view all the deployed projects on the cloud and monitor the cloud space's various resources and utilization on the Deployments Management interface. Deployment details consist of *three key areas*:

1. **Stack Events:** User can view AWS events generation during a *stack* creation. The *stack* consists of a list of tasks performed sequentially, helping the logged-in user track the status of a stack creation and update it. In the case of real-time events coming from the cloud environment, the same is available for a logged-in user to view. SNS topic notification is configured to fetch cloud-generated events.
2. **Resources:** User can view all the resources (list and graphical view) which are part of the stack, including VPC, subnet, security group, RT, NACL, etc.
3. **Chef Events:** User can view events generated during the execution of node bootstrapping and cookbook run. Same as stack events, in the case of real events generation, events are available to view for the logged-in user.

A.7 Service Management

A Service is a combination of cookbooks and scripts to run. The logged-in user can view, create, and modify/delete the services.

1. The logged in user can delete a service. Deleting a Service *logically* deletes it without a *hard delete*.

2. The following actions are required to create a Service:

   (a) A predefined set of categories (tabs) are shown to the logged-in users such as Analytics, Ingestion, Data Lake, Data Store, Enablement, and Monitoring.

   (b) The logged in user can select a category to create a service.

   (c) Clone the existing service as a template or opt for the default template to create a new service.

   (d) Provide a unique name for a service. The system checks if the name already exists and alert the logged-in user.

   (e) Add/remove the components of a service.

   (f) Key in or select the available options for each component of a service.

   (g) Add/remove the Chef cookbooks.

   (h) Add/remove the scripts.

   (i) Order the sequence of the selected cookbooks and/or scripts.

   (j) Pass the parameter to the cookbook/scripts to override the default values.

   (k) Show/hide the advanced view of the schema.

3. List the services with pagination.

4. Search for a service in a search box.

A.8 Management screens

The logged-in user can view, create, modify and delete the static data like Service Category, AMI Ids, Scripts, and AWS resource types. Also, a user can view the subnet's *security groups* and *environment types*.

1. For creating a Service Category, the logged-in user requires to key in the following fields:
   (a) Input category name which should be unique, else, the system alerts the logged-in user.
   (b) Select the subnet from a dropdown.
   (c) Select a security group from a dropdown.
   (d) Input the description.
2. Creating an AMI ID requires the following fields:
   (a) A unique AMI ID; otherwise, the system alerts the logged-in user.
   (b) Name
   (c) A category from a dropdown.
   (d) A description.
3. Creating a script requires the following fields:
   (a) Name–should be unique; otherwise, the system alerts the logged-in user.
   (b) Select a *type* from a drop-down: user data/prescript/postscript.
   (c) If *type* is *user-data*, it requires a custom source snippet or an absolute path for the source code snippet.
   (d) A description.
4. Creating the AWS resource template requires the following fields:
   (a) A unique Resource Type A; otherwise, the system alerts the logged-in user.

(b) Template name.

(c) Code snippet.

(d) Description.

A.9 Cloud Deployment Workflow

Regular users with *manager* or *admin* role can preview a project submitted for approval. Projects that are approved triggers a notification sent to the logged-in user-submitted the project for approval. Approved projects are allowed to be deployed. The entire workflow from request submission in the UI to deployment completion is described in Figure A.1. The logged-in user first submits a request using a web user interface, which calls a *Spring REST Web Services* [255]. The requests are saved into MongoDB in the form of JSON documents. Requests are simultaneously sent to *Redis*, an open-source in-memory key-value database. Subsequently, requests are submitted to a number of *AWS Chef* [166, 213, 221] worker processes to automate infrastructure in the cloud environment. As shown in the Figure A.1, the process automates creating a *Talend* cluster and a *Tableau* cluster in its own subnets.

A.10 Role Based Access Control and Active Directory Server Integration

Users are created in Microsoft Active Directory (AD) server and consist of *roles and privileges*. Each role has a set of permissions. Permissions are in the form create_user, read_user, update_sku, update_service, and others. The following Table A.1 provides details about the roles and permissions matrix.

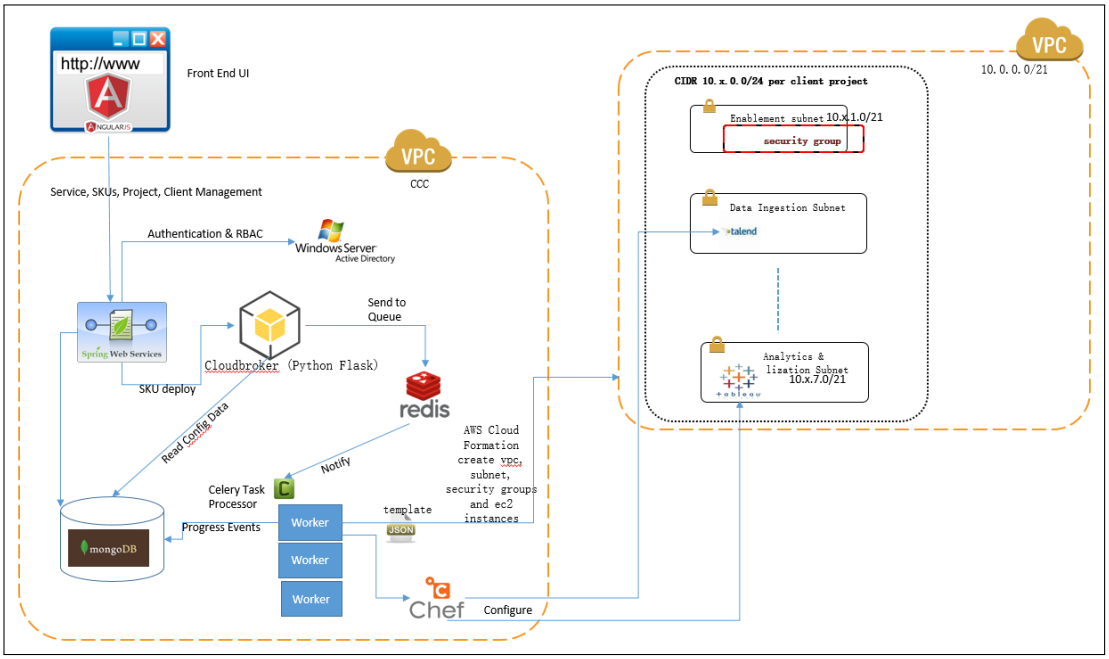


Figure A.1: End-to-end workflow for a project submission to deployment.

Table A.1: Role Based Access Control

	Admin	Manager	Normal	Support
Create Service Category	Y			
Create AMI_id	Y			
Create Script	Y			
Create AWS Template	Y			
Create Users	Y	Y		
Create Services	Y	Y		
Create SKUs	Y	Y		
Create Clients	Y	Y		
Create Projects	Y	Y	Y	
View Projects	Y	Y	Y	Y
View Reports	Y	Y	Y	Y
Create Reports	Y	Y		
Create VPC	Y	Y		
Create Logs	Y	Y		

A.11 Logs, Reports, and Backups

Using the logs, reports, and backups interface, the logged-in user can view the API and cloud broker's log data. The interface provides an option to choose the number of records of log data the logged-in user intends to see for API or Cloud broker. Also, the logged-in user can view the audit log for a period of time by choosing a start and end date. Reports interface is used for viewing the summary and relationship of user/PROJECT/VPC for the current development cycle. The interface is designed as a simple table to view and available to download the PDF/CSV for report purpose. As a backup mechanism for security, data (in MongoDB) is automatically backed up on the cloud.

A.12 Proposed Architecture for an Automated Cloud Deployment

The proposed architecture for an automated cloud deployment is shown in Figure A.2. Each of the *5 components* are described below:

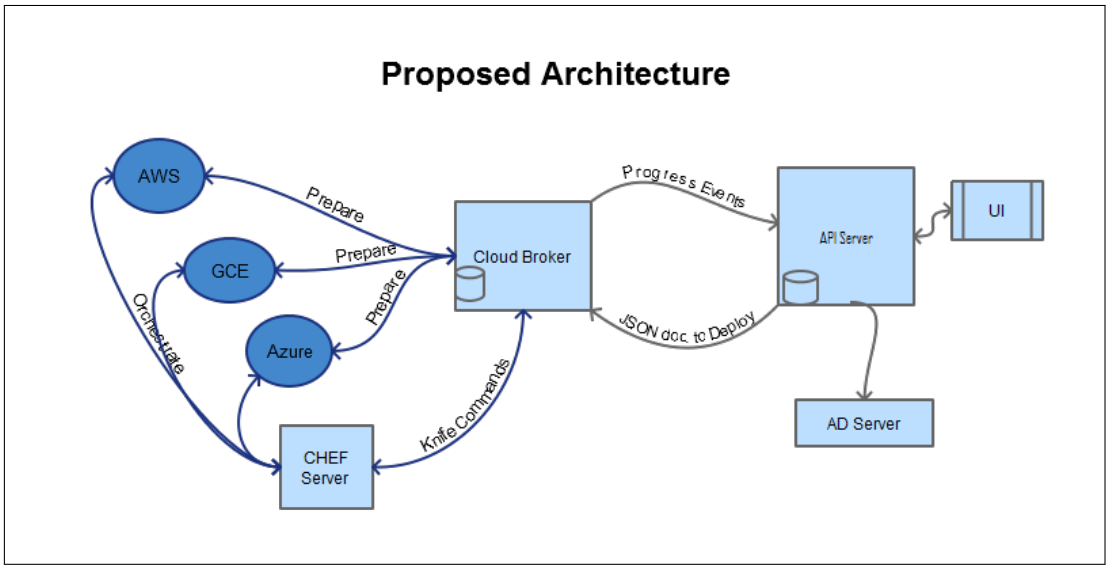


Figure A.2: Automated cloud deployment architecture consists of *five components*–UI, API Server, AD Server, Cloud Broker and Chef Server

A.12.1 Components

Essentially, five principal components make up the system: UI, API Server, AD Server, Cloud Broker, and Chef Server. The components are described as follows.

UI

The UI Layer is responsible for showing a view based on *Role-Based Access Control*, specified by the API server. CRUD operations are provided at this layer for SKU, Services, Users, Projects, Service Category, AMI IDs, Scripts, and AWS Resource Templates.

API Server

The API Server is responsible for storing and serving users, projects, SKU, and other models. Data is stored in MongoDB. The API Server is responsible for generating a JSON manifest file that abstracts Cloud resources required for deployment to the chosen cloud provider. The API server also provides endpoints to manipulate information in the Cloud Broker. The API Server can publish progress events that it receives from the cloud broker. Authentication for API endpoint access is verified using an Active Directory server.

Active directory (AD)

Active Directory [117] allows authorized internal and external users to access this web interface.

Cloud Broker

The broker component [67, 127] is responsible for translating the JSON manifest from the API server and preparing the cloud provider environment. the Cloud Broker hosts an instance of MongoDB to store Chef/Knife Commands, SSH Keys, Logging information, and other data. Cloud Broker can publish events to authenticated subscribers. Cloud Broker has API endpoints to retrieve deployment information, its progress and can terminate deployed resources. It is also re-

sponsible for initiating knife commands to Chef Server to execute orchestration source code stored as Cookbooks with Recipes.

Chef Server

The *Chef* is a continuous automation platform for cloud infrastructure provisioning. Clusters created by *Chef* are continuously verified to ensure any patching and configuration update is applied to all nodes in the cluster. The Chef works with any cloud provider as well as in-house deployments. Cookbooks and recipes can exist on the *Chef* server and are displayed on the *user interface* to be viewed and associated with services.

A.13 Summary

This chapter presented an automation framework to create and configure a *big data cluster* into a *cloud environment*. The automated cloud provisioning helped saving time on each occasion, a custom cluster environment (installation and configuration) is created in a cloud environment. For a cloud instance, ACD can create a multi-node Hadoop cluster and get several other applications installed within a few minutes by selecting desired components and their specifications. ACD consists of a *User Management* console to create, update, delete users. Each user logs in to the ACD with different levels of privileges. Users' details are linked with the *Microsoft Active Directory* for *Role Based Access Control*. *Deployment Management* allows managing a list of previously deployed *projects* in the cloud. Using *Service Management* component, the logged-in user can manage the *Chef* and other custom scripts that are executed in the cloud environment. *Services* can be updated from a *Management Screen*. List of services are combined together to create a *SKU*, and several *SKUs* forms a *Project*.

Bibliography

[1] https://gdc.cancer.gov/. Accessed: 1 jun, 2019.

[2] https://www.adobe.com/in/analytics/adobe-analytics.html. Accessed: 1 Aug, 2019.

[3] https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html. Accessed: 27 Oct. 2018.

[4] https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining. Accessed: 1 Aug, 2019.

[5] https://data.world/promptcloud/product-details-on-flipkart-com. Accessed: 27 Oct. 2018.

[6] https://data.world/promptcloud/fashion-products-on-amazon-com. Accessed: 1 April. 2021.

[7] https://spark.apache.org/docs/latest/mllib-clustering.html. Accessed: 27 June, 2018.

[8] https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlAboutDataConsistency.html. Accessed: 1 Oct, 2018.

[9] https://jena.apache.org/. Accessed: 1 jun, 2019.

[10] https://gdc.cancer.gov/. Accessed: 1 jun, 2019.

[11] https://spark.apache.org/docs/latest/mllib-clustering.html. Accessed: 27 Oct. 2018.

[12] https://spark.apache.org/docs/latest/mllib-clustering.html. Accessed: 27 Oct. 2018.

[13] https://spark.apache.org/docs/2.2.0/mllib-statistics.html#stratified-sampling. Accessed: 22 Jan. 2019.

[14] https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/ml/classification/RandomForestClassificationl Accessed: 22 Jan. 2019.

[15] https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/ml/evaluation/MulticlassClassificationEvaluat Accessed: 22 Jan. 2019.

[16] https://gdc.cancer.gov/. Accessed: 1 jun, 2019.

[17] https://splunkbase.splunk.com/app/2890/. Accessed: 2 Feb. 2019.

[18] https://splunkbase.splunk.com/. Accessed: 2 Feb. 2019.

[19] https://www.linkedin.com/pulse/flume-kafka-real-time-event-processing-lan-jiang/. Accessed: 27 Oct. 2018.

[20] https://grouplens.org/datasets/movielens/100k/. Accessed: 27 Oct. 2018.

[21] https://data.world/promptcloud/product-details-on-flipkart-com. Accessed: 27 Oct. 2018.

[22] https://github.com/PacktPublishing/Splunk-Operational-Intelligence-Cookbook-Second-Edition/find/master. Accessed: 27 Oct. 2018.

[23] Kaggle news category dataset. https://www.kaggle.com/rmisra/news-category-dataset/kernels. Accessed: 1 Sep. 2020.

[24] Current employment statistics - ces (national). https://www.bls.gov/web/empsit/ceseesummary.htm, type = Web Page, 2020.

[25] Data on covid-19 (coronavirus) by our world in data. https://github.com/owid/covid-19-data/tree/master/public/data, 2020.

[26] Drive strategic decision-making with social media analytics, sprout social. https://sproutsocial.com/, 2020.

[27] News coverage of coronavirus in 2020 is very different than it was for ebola in 20. https://time.com/5779872/coronavirus-ebola-news-coverage/, 2020.

[28] Social media monitoring tools and services report 2018, 9th edition, ideya inc, 2020. Accessed:1 August. 2020.

[29] Template:covid-19 pandemic data. https://en.wikipedia.org/wiki/Template:COVID-19_pandemic_data, 2020.

[30] Common objects in context. https://cocodataset.org/#home, 2021.

[31] Media coverage of the covid-19 pandemic. https://en.wikipedia.org/wiki/Media_coverage_of_the_COVID-19_pandemic, 2021.

[32] Tlc trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2021.

[33] Hervé Abdi and Paul Molin. Lilliefors/van soestâĂŹs test of normality. *Encyclopedia of measurement and statistics*, pages 540–544, 2007.

[34] Deepak K Agarwal and Bee-Chung Chen. *Statistical methods for recommender systems*. Cambridge University Press, New York, 2016.

[35] K.D. Agarwal and B. Chen. chapter 7. In *Statistical Methods for Recommender Systems*. Cambridge University Press, 2015.

[36] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.

[37] Tyler Akidau, Reuven Lax, and Slava Chernyak. *chapter 1*. O'Reilly Media, 2018.

[38] Rizik MH Al-Sayyed, Wadi'A Hijawi, Anwar M Bashiti, Ibrahim AlJarah, Nadim Obeid, and Omar Y Adwan. An investigation of microsoft azure and amazon web services from users' perspectives. *International Journal of Emerging Technologies in Learning*, 14(10), 2019.

[39] Mehdi Assefi, Ehsun Behravesh, Guangchi Liu, and Ahmad P Tafti. Big data machine learning using apache spark mllib. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3492–3498. IEEE, 2017.

[40] Krassimir Atanassov, Eulalia Szmidt, Janusz Kacprzyk, and Vassia Atanassova. An approach to a constructive simplification of multiagent multicriteria decision making problems via intercriteria analysis. *Comptes rendus de l'Académie bulgare des Sciences*, 70(8):1147–1157, 2017.

[41] Pradeep K Atrey, M Anwar Hossain, Abdulmotaleb El Saddik, and Mohan S Kankanhalli. Multimodal fusion for multimedia analysis: a survey. *Multimedia systems*, 16(6):345–379, 2010.

[42] John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. Semantic-based automated reasoning for aws access policies using smt. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–9. IEEE, 2018.

[43] Mihal Badjonski, Mirjana Ivanović, and Zoran Budimac. Agent oriented programming language lass. In *Object-oriented technology and computing systems re-engineering*, pages 111–121. Horwood Publishing, Ltd., 1999.

[44] Vinit Kumar Baheti. Windows azure hdinsight: where big data meets the cloud. In *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pages 1–2. IEEE, 2014.

[45] S. Bansal, C. Gupta, and A. Sinha. Clickstream & behavioral analysis with context awareness for e-commercial applications. In *2017 Tenth International Conference on Contemporary Computing (IC3)*, pages 1–6.

[46] A. Batyuk and V. Voityshyn. Apache storm based on topology for real-time processing of streaming data from social networks. In *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*, pages 345–349, 2018.

[47] A. Batyuk and V. Voityshyn. Apache storm based on topology for real-time processing of streaming data from social networks. In *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*, pages 345–349, 2018.

[48] A. Batyuk and V. Voityshyn. Apache storm based on topology for real-time processing of streaming data from social networks. In *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*, pages 345–349, 2018.

[49] Marouane Birjali, Abderrahim Beni-Hssane, and Mohammed Erritali. Analyzing social media through big data using infosphere biginsights and apache flume. *Procedia computer science*, 113:280–285, 2017.

[50] J Martin Bland and Douglas G Altman. Survival probabilities (the kaplan-meier method). *Bmj*, 317(7172):1572–1580, 1998.

[51] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[52] Savong BOU, Hiroyuki KITAGAWA, and Toshiyuki AMAGASA. Cbix: Incremental sliding-window aggregation for real-time analytics over out-of-order data streams, 2018.

[53] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*, pages 89–96, 2005.

[54] David Carasso. *Exploring Splunk*. CITO Research New York, USA, 2012.

[55] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in apache flink®: consistent stateful distributed stream processing. *Proceedings of the VLDB Endowment*, 10(12):1718–1729, 2017.

[56] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[57] Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr, and Tom M Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110. ACM, 2010.

[58] Jeff Carpenter and Eben Hewitt. *Cassandra: the definitive guide: distributed data at web scale*. O'Reilly Media, 2020.

[59] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[60] Avkash Chauhan, Valentine Fontama, Michele Hart, Wee-Hyong Tok, and Buck Woody. *Introducing Microsoft Azure HDInsight*. Microsoft press, 2014.

[61] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[62] Jianguo Chen, Kenli Li, Zhuo Tang, Kashif Bilal, Shui Yu, Chuliang Weng, and Keqin Li. A parallel random forest algorithm for big data in a spark cloud computing environment. *IEEE Transactions on Parallel & Distributed Systems*, (1):1–1, 2017.

[63] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[64] Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong learning for sentiment classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 750–756, 2015.

[65] Arnaud Cogoluegnes, Thierry Templier, Gary Gregory, and Olivier Bazoud. *Spring batch in action*. Manning Publications Co., 2011.

[66] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[67] Frédéric Desprez and Jonathan Rouzaud-Cornabas. Simgrid cloud broker: Simulating the amazon aws cloud. 2013.

[68] Rahul Dey and Fathi M Salemt. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[69] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[70] Thomas W Dinsmore. Streaming analytics. In *Disruptive analytics*, pages 117–144. Springer, 2016.

[71] M. A. Domingues, C. V. Sundermann, M. G. Manzato, R. M. Marcacini, and S. O. Rezende. Exploiting text mining techniques for contextual recommendations. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 2, pages 210–217.

[72] M. A. Domingues, C. V. Sundermann, M. G. Manzato, R. M. Marcacini, and S. O. Rezende. Exploiting text mining techniques for contextual recommendations. In *2014*

IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), volume 2, pages 210–217.

[73] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. Multimodal interfaces: A survey of principles, models and frameworks. In *Human machine interaction*, pages 3–26. Springer, 2009.

[74] Ted Dunning and Ellen Friedman. *Streaming architecture: new designs using Apache Kafka and MapR streams*. " O'Reilly Media, Inc.", 2016.

[75] SD Erokhin. A review of scientific research on artificial intelligence. In *2019 Systems of Signals Generating and Processing in the Field of on Board Communications*, pages 1–4. IEEE, 2019.

[76] William P Eveland Jr and Dhavan V Shah. The impact of individual and interpersonal factors on perceived news media bias. *Political psychology*, 24(1):101–117, 2003.

[77] M. R. Hoseiny Farahabady, H. R. Dehghani Samani, Y. Wang, A. Y. Zomaya, and Z. Tari. A qos-aware controller for apache storm. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 334–342, Oct 2016.

[78] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *European conference on computer vision*, pages 15–29. Springer, 2010.

[79] Golnoosh Farnadi, Jie Tang, Martine De Cock, and Marie-Francine Moens. User profiling through deep multimodal fusion. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 171–179, 2018.

[80] Geli Fei, Shuai Wang, and Bing Liu. Learning cumulatively to become more knowledgeable. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1565–1574, New York, NY, USA, 2016. ACM.

[81] Simran Fitzgerald, George Mathews, Colin Morris, and Oles Zhulyn. Using nlp techniques for file fragment classification. *Digital Investigation*, 9:S44–S49, 2012.

[82] Ellen Friedman and Kostas Tzoumas. *Introduction to Apache Flink: stream processing for real time and beyond*. " O'Reilly Media, Inc.", 2016.

[83] Andrea Galeotti, Sanjeev Goyal, Matthew O Jackson, Fernando Vega-Redondo, and Leeat Yariv. Network games. *The review of economic studies*, 77(1):218–244, 2010.

[84] Nishant Garg. *Apache Kafka*. Packt Publishing Ltd, 2013.

[85] Lars George. *HBase: the definitive guide: random access to your planet-size data*. " O'Reilly Media, Inc.", 2011.

[86] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[87] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

[88] Kannan Govindan, Sivakumar Rajendran, Joseph Sarkis, and Parasurama Murugesan. Multi criteria decision making approaches for green supplier evaluation and selection: a literature review. *Journal of Cleaner Production*, 98:66–83, 2015.

[89] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Foundations of computer science, 2000. proceedings. 41st annual symposium on*, pages 359–366. IEEE, 2000.

[90] Darren Guinness, Edward Cutrell, and Meredith Ringel Morris. Caption crawler: Enabling reusable alternative text descriptions using reverse image search. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2018.

[91] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. Mapreduce in the clouds for science. In *2010 IEEE second international conference on cloud computing technology and science*, pages 565–572. IEEE, 2010.

[92] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2014.

[93] R. D. Hagan, C. A. Phillips, M. A. Langston, and B. J. Rhodes. Classification and anomaly detection in traffic patterns of new york city taxis: A case study in compound analytics. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1169–1174, May 2018.

[94] R. Hanamanthrao and S. Thejaswini. Real-time clickstream data analytics and visualization. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2139–2144.

[95] R. Hanamanthrao and S. Thejaswini. Real-time clickstream data analytics and visualization. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2139–2144.

[96] Ramanna Hanamanthrao and S Thejaswini. Real-time clickstream data analytics and visualization. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2139–2144. IEEE, 2017.

[97] M. Hanif, H. Yoon, S. Jang, and C. Lee. An adaptive sla-based data flow mechanism for stream processing engines. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 81–86.

[98] M. Hanif, H. Yoon, S. Jang, and C. Lee. An adaptive sla-based data flow mechanism for stream processing engines. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 81–86.

[99] M. Hanif, H. Yoon, S. Jang, and C. Lee. An adaptive sla-based data flow mechanism for stream processing engines. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 81–86.

[100] M. Hanif, H. Yoon, S. Jang, and C. Lee. An adaptive sla-based data flow mechanism for stream processing engines. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 81–86.

[101] Michael Hausenblas and Nathan Bijnens. Lambda architecture. *URL: http://lambda-architecture. net/. Luettu*, 6:2014, 2015.

[102] J. Heidrich, A. Trendowicz, and C. Ebert. Exploiting big data's benefits. *IEEE Software*, 33(4):111–116, 2016.

[103] J. Heidrich, A. Trendowicz, and C. Ebert. Exploiting big data's benefits. *IEEE Software*, 33(4):111–116, 2016.

[104] J. Heidrich, A. Trendowicz, and C. Ebert. Exploiting big data's benefits. *IEEE Software*, 33(4):111–116, 2016.

[105] Eben Hewitt. *Cassandra: the definitive guide*. " O'Reilly Media, Inc.", 2010.

[106] Erik G Hoel, Wee-Liang Heng, and Dale Honeycutt. High performance multimodal networks. In *International symposium on spatial and temporal databases*, pages 308–327. Springer, 2005.

[107] Steve Hoffman. *Apache Flume: distributed log collection for Hadoop*. Packt Publishing Ltd, 2013.

[108] Steve Hoffman. *Apache flume: Distributed log collection for hadoop*. Packt Publishing Ltd, 2015.

[109] Xianbin Hong, Gautam Pal, Sheng-Uei Guan, Prudence Wong, Dawei Liu, Ka Lok Man, and Xin Huang. Semi-unsupervised lifelong learning for sentiment classification: Less manual

data annotation and more self-studying. In *Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference*, HPCCT 2019, pages 87–92, New York, NY, USA, 2019. ACM.

[110] Xianbin Hong, Prudence Wong, Dawei Liu, Sheng-Uei Guan, Ka Lok Man, and Xin Huang. Lifelong machine learning: Outlook and direction. In *Proceedings of the 2nd International Conference on Big Data Research*, pages 76–79. ACM, 2018.

[111] Badr Hssina, Abdelkarim Merbouha, Hanane Ezzikouri, and Mohammed Erritali. A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2), 2014.

[112] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272.

[113] Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):116–142, 2004.

[114] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 8, 2014.

[115] A. Ichinose, A. Takefusa, H. Nakada, and M. Oguchi. A study of a video analysis framework using kafka and spark streaming. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2396–2401.

[116] A. Ichinose, A. Takefusa, H. Nakada, and M. Oguchi. A study of a video analysis framework using kafka and spark streaming. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2396–2401.

[117] David Iseminger. *Active directory services for Microsoft windows 2000*. Microsoft Press, 1999.

[118] Mohammad Kamrul Islam and Aravind Srinivasan. *Apache Oozie: The Workflow Scheduler for Hadoop*. " O'Reilly Media, Inc.", 2015.

[119] Kevin Jacobs and Kacper Surdy. Apache flink: Distributed stream data processing. Technical report, 2016.

[120] Darryl C Jarman, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic testing for adobe data analytics software. In *Proceedings of the 2nd International Workshop on Metamorphic Testing*, pages 21–27. IEEE Press, 2017.

[121] M Tim Jones. Process real-time big data with twitter storm. *IBM Technical Library*, 2013.

[122] Cengiz Kahraman. *Fuzzy multi-criteria decision making: theory and applications with recent developments*, volume 16. Springer Science & Business Media, 2008.

[123] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[124] Monisha Kanakaraj and Ram Mohana Reddy Guddeti. Nlp based sentiment analysis on twitter data using ensemble classifiers. In *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–5. IEEE, 2015.

[125] Ziya Karakaya, Ali Yazici, and Mohammed Alayyoub. A comparison of stream processing frameworks. In *2017 International Conference on Computer and Applications (ICCA)*, pages 1–12. IEEE, 2017.

[126] C. Kaushal and H. Singh. Comparative study of recent sequential pattern mining algorithms on web clickstream data. In *2015 IEEE Power, Communication and Information Technology Conference (PCITC)*, pages 652–656.

[127] Prashant Khanna, Sonal Jain, and BV Babu. Cloud broker: Working in federated structures. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1273–1278. IEEE, 2014.

[128] F. Ben Kharrat, A. Elkhleifi, and R. Faiz. Recommendation system based contextual analysis of facebook comment. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–6.

[129] F. Ben Kharrat, A. Elkhleifi, and R. Faiz. Recommendation system based contextual analysis of facebook comment. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–6.

[130] H. Kim, S. Madhvanath, and T. Sun. Hybrid active learning for non-stationary streaming data with asynchronous labeling. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 287–292.

[131] H. Kim, S. Madhvanath, and T. Sun. Hybrid active learning for non-stationary streaming data with asynchronous labeling. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 287–292.

[132] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[133] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792. IEEE, 2015.

[134] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792. IEEE, 2015.

[135] R. Klapsing, G. Neumann, and W. Conen. Semantics in web engineering: applying the resource description framework. *IEEE MultiMedia*, 8(2):62–68, April 2001.

[136] Myoung-Wan Koo, Chin-Hui Lee, and Biing-Hwang Juang. Speech recognition and utterance verification based on a generalized confidence score. *IEEE Transactions on Speech and Audio Processing*, 9(8):821–832, 2001.

[137] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[138] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

[139] K. R. Kurte, S. S. Durbha, R. L. King, N. H. Younan, and R. Vatsavai. Semantics-enabled framework for spatial image information mining of linked earth observation data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(1):29–44, Jan 2017.

[140] Chuck Lam. *Hadoop in action*. Manning Publications Co., 2010.

[141] E Larios, Y Zhang, K Yan, Z Di, S LeDévédec, F Groffen, and Fons J Verbeek. Automation in cytomics: a modern rdbms based platform for image analysis and management in high-throughput screening experiments. In *International Conference on Health Information Science*, pages 76–87. Springer, 2012.

[142] M. K. LeblebicÃőoÄ§lu, Y. Zengin, and K. W. Schmidt. A new multi-agent decision making structure and application to model-based fault diagnosis problem. In *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

[143] C. H. Lee and C. Y. Lin. Implementation of lambda architecture: A restaurant recommender system over apache mesos. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 979–985.

[144] C. H. Lee and C. Y. Lin. Implementation of lambda architecture: A restaurant recommender system over apache mesos. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 979–985.

[145] Yeonhee Lee and Youngseok Lee. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review*, 43(1):5–13, 2013.

[146] Dingcheng Li, Cory M Endle, Sahana Murthy, Craig Stancl, Dale Suesse, Davide Sottara, Stanley M Huff, Christopher G Chute, and Jyotishman Pathak. Modeling and executing electronic health records driven phenotyping algorithms using the nqf quality data model and jboss® drools engine. In *AMIA annual symposium proceedings*, volume 2012, page 532. American Medical Informatics Association, 2012.

[147] G. Li, M. Chi, and G. Pal. Expert cf: Sparse data matrix completion with artificial experts. *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, 6(1):p20–22, Oct, 2017.

[148] Lianghao Li and Qiang Yang. Lifelong machine learning test. In *Proceedings of the Workshop on âĂIJBeyond the Turing TestâĂİ of AAAI Conference on Artificial Intelligence*, 2015.

[149] Mei Li, Shuang Liu, and Zhong Zhang. Deep tensor fusion network for multimodal ground-based cloud classification in weather station networks. *Ad Hoc Networks*, 96:101991, 2020.

[150] Shen Li, Paul Gerver, John MacMillan, Daniel Debrunner, William Marshall, and Kun-Lung Wu. Challenges and experiences in building an efficient apache beam runner for ibm streams. *Proceedings of the VLDB Endowment*, 11(12):1742–1754, 2018.

[151] Weixin Li, Jungseock Joo, Hang Qi, and Song-Chun Zhu. Joint image-text news topic detection and tracking by multimodal topic and-or graph. *IEEE Transactions on Multimedia*, 19(2):367–381, 2016.

[152] Hubert W Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American statistical Association*, 62(318):399–402, 1967.

[153] Jimmy Lin. The lambda and the kappa. *IEEE Internet Computing*, 21(5):60–66, 2017.

[154] Yu-Ru Lin, James P Bagrow, and David Lazer. More voices than ever? quantifying media bias in networks. *arXiv preprint arXiv:1111.1227*, 2011.

[155] Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan, and Alex C Kot. Global context-aware attention lstm networks for 3d action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1647–1656, 2017.

[156] Qian Liu, Bing Liu, Yuanlin Zhang, Doo Soon Kim, and Zhiqiang Gao. Improving opinion aspect extraction using semantic similarity and aspect associations. In *AAAI*, pages 2986–2992, 2016.

[157] Z. Liu, Y. Wang, M. Dontcheva, M. Hoffman, S. Walker, and A. Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):321–330, Jan 2017.

[158] Q. Long and S. Li. The innovation network as a complex adaptive system: Flexible multi-agent based modeling, simulation, and evolutionary decision making. In *2014 Fifth International Conference on Intelligent Systems Design and Engineering Applications*, pages 1060–1064.

[159] Guangyi Lv, Shuai Wang, Bing Liu, Enhong Chen, and Kun Zhang. Sentiment classification by leveraging the shared knowledge from a sequence of domains. In *International Conference on Database Systems for Advanced Applications*, pages 795–811. Springer, 2019.

[160] Francois Maas, Gerard ; Garillot. Learning spark streaming. In *Learning Spark Streaming*, book section Chapter 3: Streaming Application Design. O'Reilly Media, Inc., 2018.

[161] Michael Malak and Robin East. *Spark GraphX in action*. Manning Publications Co., 2016.

[162] Amit Mandelbaum and Daphna Weinshall. Distance-based confidence score for neural network classifiers. *arXiv preprint arXiv:1709.09844*, 2017.

[163] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[164] M. A. Manzoor and Y. Morgan. Real-time support vector machine based network intrusion detection system using apache storm. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–5, Oct 2016.

[165] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. Explain images with multimodal recurrent neural networks. *arXiv preprint arXiv:1410.1090*, 2014.

[166] Matthias Marschall. *Chef infrastructure automation cookbook*. Packt Publishing Ltd, 2015.

[167] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. New York; Manning Publications Co., 2015.

[168] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[169] Diana Maynard, David Dupplaw, and Jonathon Hare. Multimodal sentiment analysis of social media. 2013.

[170] Andrew McCallum and Kamal Nigam. Text classification by bootstrapping with keywords, em and shrinkage. *Unsupervised Learning in Natural Language Processing*, 1999.

[171] N Madurai Meenachi and M Sai Baba. Web ontology language editors for semantic web-a survey. *International Journal of Computer Applications*, 53(12), 2012.

[172] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[173] Z. Miao and L. Fan. A novel multi-agent decision making architecture based on dual's dual problem formulation. *IEEE Transactions on Smart Grid*, PP(99):1–1, 2016.

[174] Vivek Mishra. *Beginning Apache Cassandra Development*. Apress, 2014.

[175] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, B Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[176] Martin Möhle. A convergence theorem for markov chains arising in population genetics and the coalescent with selfing. *Advances in Applied Probability*, 30(2):493–512, 1998.

[177] Seungwhan Moon, Leonardo Neves, and Vitor Carvalho. Multimodal named entity disambiguation for noisy social media posts. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2000–2008, 2018.

[178] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[179] John N Mordeson and Peng Chang-Shyh. Operations on fuzzy graphs. *Information sciences*, 79(3-4):159–170, 1994.

[180] Vineeth G Nair. *Getting Started with Beautiful Soup*. Packt Publishing Ltd, 2014.

[181] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[182] Temitope B Oriola and W Andy Knight. Covid-19, george floyd and human security, 2020.

[183] G. Pal, G. Li, and K. Atkinson. Big data real time ingestion and machine learning. In *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*, pages 25–31, Aug 2018.

[184] G. Pal, G. Li, and K. Atkinson. Big data real time ingestion and machine learning. In *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*, pages 25–31, Aug 2018.

[185] G. Pal, G. Li, and K. Atkinson. Multi-agent item to item contextual big data recommender system. *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, 6(1):p58–59, Oct, 2017.

[186] G. Pal, G. Li, and K. Atkinson. Multi-agent item to item contextual big data recommender system. *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, 6(1):p58–59, Oct, 2017.

[187] G. Pal, G. Li, and K. Atkinson. Multi-agent item to item contextual big data recommender system. *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, 6(1):p58–59, Oct, 2017.

[188] Gautam Pal, Gangmin Li, and Katie Atkinson. Big data ingestion and lifelong learning architecture. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5420–5423. IEEE, 2018.

[189] Gautam Pal, Gangmin Li, and Katie Atkinson. Big data real time ingestion and machine learning. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pages 25–31. IEEE, 2018.

[190] Gautam Pal, Gangmin Li, and Katie Atkinson. Multi-agent big-data lambda architecture model for e-commerce analytics. *Data*, 3(4):58, 2018.

[191] Gautam Pal, Gangmin Li, and Katie Atkinson. Multi-agent big-data lambda architecture model for e-commerce analytics. *Data*, 3(4):58, 2018.

[192] Mrutyunjaya Panda and Manas Ranjan Patra. Network intrusion detection using naive bayes. *International journal of computer science and network security*, 7(12):258–263, 2007.

[193] Manos Papagelis, Dimitris Plexousakis, and Themistoklis Kutsuras. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *International Conference on Trust Management*, pages 224–239. Springer, 2005.

[194] Wen-Chih Peng and Zhung-Xun Liao. Mining sequential patterns across multiple sequence databases. *Data & Knowledge Engineering*, 68(10):1014–1033, 2009.

[195] Arthur V Peterson Jr. Expressing the kaplan-meier estimator as a function of empirical subsurvival functions. *Journal of the American Statistical Association*, 72(360a):854–858, 1977.

[196] Makan Pourzandi, Mohamed Fekih Ahmed, Mohamed Cheriet, and Chamseddine Talhi. Multi-tenant isolation in a cloud environment using software defined networking, March 6 2018. US Patent 9,912,582.

[197] Mark Proctor. Drools: a rule engine for complex event processing. In *Proceedings of the 4th international conference on Applications of Graph Transformations with Industrial Relevance*, pages 2–2. Springer-Verlag, 2011.

[198] Robert D Purves. Optimum numerical integration methods for estimation of area-under-the-curve (auc) and area-under-the-moment-curve (aumc). *Journal of pharmacokinetics and biopharmaceutics*, 20(3):211–226, 1992.

[199] Tingting Qiao, Jing Zhang, Duanqing Xu, and Dacheng Tao. Mirrorgan: Learning text-to-image generation by redescription. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1505–1514, 2019.

[200] Adrian E Raftery. A model for high-order markov chains. *Journal of the Royal Statistical Society: Series B (Methodological)*, 47(3):528–539, 1985.

[201] M. M. Rahman. Contextual recommendation system. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6.

[202] M. M. Rahman. Contextual recommendation system. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6.

[203] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. New Jersey, USA, 2003.

[204] M. Rawat, N. Goyal, and S. Singh. Advancement of recommender system based on clickstream data using gradient boosting and random forest classifiers. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6.

[205] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[206] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[207] Michele Reilly and Santi Thompson. Reverse image lookup: assessing digital library users and reuses. *Journal of Web Librarianship*, 11(1):56–68, 2017.

[208] Y. Ren, M. Tomko, F. D. Salim, J. Chan, C. Clarke, and M. Sanderson. A location-query-browse graph for contextual recommendation. *IEEE Transactions on Knowledge and Data Engineering*, PP(99):1–1, 2017.

[209] Y. Ren, M. Tomko, F. D. Salim, J. Chan, C. Clarke, and M. Sanderson. A location-query-browse graph for contextual recommendation. *IEEE Transactions on Knowledge and Data Engineering*, PP(99):1–1, 2017.

[210] R. Reshma, G. Ambikesh, and P. S. Thilagam. Alleviating data sparsity and cold start in recommender systems using social behaviour. In *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1–8.

[211] R. Reshma, G. Ambikesh, and P. S. Thilagam. Alleviating data sparsity and cold start in recommender systems using social behaviour. In *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1–8.

[212] Laura Rettig, Mourad Khayati, Philippe Cudré-Mauroux, and Michal Piórkowski. Online anomaly detection over big data streams. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1113–1122. IEEE, 2015.

[213] Todd Rosner. *Learning AWS OpsWorks*. Packt Publishing Ltd, 2013.

[214] Salvatore Ruggieri. Efficient c4. 5 [classification algorithm]. *IEEE transactions on knowledge and data engineering*, 14(2):438–444, 2002.

[215] Paul Ruvolo and Eric Eaton. ELLA: An efficient lifelong learning algorithm. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 507–515, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[216] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515, 2013.

[217] Mauricio Salatino, Mariano De Maio, and Esteban Aliverti. *Mastering jboss drools 6*. Packt Publishing Ltd, 2016.

[218] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1(3-4):145–164, 2016.

[219] Kamal Sarkar, Mita Nasipuri, and Suranjan Ghose. Machine learning based keyphrase extraction: Comparing decision trees, naïve bayes, and artificial neural networks. *JIPS*, 8(4):693–712, 2012.

[220] Michael Scholz et al. R package clickstream: analyzing clickstream data with markov chains. *Journal of Statistical Software*, 74(4):1–17, 2016.

[221] Adam Shackelford. Working with aws opsworks. In *Beginning Amazon Web Services with Node. js*, pages 31–59. Springer, 2015.

[222] Z. Sharifi, M. Rezghi, and M. Nasiri. A new algorithm for solving data sparsity problem based-on non negative matrix factorization in recommender systems. In *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 56–61.

[223] Z. Sharifi, M. Rezghi, and M. Nasiri. A new algorithm for solving data sparsity problem based-on non negative matrix factorization in recommender systems. In *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 56–61.

[224] B. Shu, H. Chen, and M. Sun. Dynamic load balancing and channel strategy for apache flume collecting real-time data stream. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pages 542–549.

[225] Jonathan J Shuster. Student t-tests for potentially abnormal data. *Statistics in medicine*, 28(16):2170–2184, 2009.

[226] Erkki Siira and Vili Törmänen. The impact of nfc on multimodal social media application. In *2010 Second International Workshop on Near Field Communication*, pages 51–56. IEEE, 2010.

[227] DANIEL L SILVER. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science*, 8(2):277–294, 1996.

[228] Daniel L. Silver, Geoffrey Mason, and Lubna Eljabu. Consolidation using sweep task rehearsal: Overcoming the stability-plasticity problem. In Denilson Barbosa and Evangelos Milios, editors, *Advances in Artificial Intelligence*, pages 307–322, Cham, 2015. Springer International Publishing.

[229] Daniel L. Silver and Robert E. Mercer. The task rehearsal method of life-long learning: Overcoming impoverished data. In Robin Cohen and Bruce Spencer, editors, *Advances in Artificial Intelligence*, pages 90–101, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[230] Daniel L. Silver and Ryan Poirier. Sequential consolidation of learned task knowledge. In Ahmed Y. Tawfik and Scott D. Goodwin, editors, *Advances in Artificial Intelligence*, pages 217–232, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[231] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*, 2013.

[232] Maninder Pal Singh, Mohammad A Hoque, and Sasu Tarkoma. Analysis of systems to process massive data stream. *arXiv preprint arXiv:1605.09021*, 2016.

[233] S. Singh and Y. Liu. A cloud service architecture for analyzing big monitoring data. *Tsinghua Science and Technology*, 21(1):55–70, 2016.

[234] Florentin Smarandache. *Neutrosophic Theory and Its Applications, Vol. I: Collected Papers*. Infinite Study, 2014.

[235] Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan, Bhavani Thuraisingham, and Joey Burton Ingram. Spark-based anomaly detection over multi-source vmware performance data in real-time. In *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 1–8. IEEE, 2014.

[236] Mohammad Soleymani, David Garcia, Brendan Jou, Björn Schuller, Shih-Fu Chang, and Maja Pantic. A survey of multimodal sentiment analysis. *Image and Vision Computing*, 65:3–14, 2017.

[237] S. Son, S. Lee, M. S. Gil, M. J. Choi, and Y. S. Moon. Locality aware traffic distribution in apache storm for energy analytics platform. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 721–724, Jan 2018.

[238] Bhargav Srinivasa-Desikan. *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd, 2018.

[239] Dusan Stevanovic, Natalija Vlajic, and Aijun An. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing*, 13(1):698–708, 2013.

[240] T. Sun, M. Wang, and Z. Liang. Predictive modeling of potential customers based on the customers clickstream data: A field study. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 2221–2225.

[241] MS Sunitha and A Vijayakumar. Complement of a fuzzy graph. *Indian Journal of pure and applied Mathematics*, 33(9):1451–1464, 2002.

[242] Shan Suthaharan. Decision tree learning. In *Machine Learning Models and Algorithms for Big Data Classification*, pages 237–269. Springer, 2016.

[243] Madhusmita Swain, Sanjit Kumar Dash, Sweta Dash, and Ayeskanta Mohapatra. An approach for iris plant classification using neural network. *International Journal on Soft Computing*, 3(1):79, 2012.

[244] Sabine Tan, Kay L O'Halloran, Peter Wignell, Kevin Chai, and Rebecca Lange. A multimodal mixed methods approach for examining recontextualisation patterns of violent extremist images in online media. *Discourse, Context & Media*, 21:18–35, 2018.

[245] Inês Catarina Ferreira Teixeira. Event-driven real-time streaming approach for big data, applied to an end-to-end supply chain. 2019.

[246] R Thottuvaikkatumana. Data modeling considerations. In *Cassandra Design Patterns*. Packt Publishing, 2nd edition, 2015.

[247] S. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, Boston, MA, 1996.

[248] S. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, Boston, MA, 1996.

[249] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[250] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th international conference on data engineering (ICDE 2010)*, pages 996–1005. IEEE, 2010.

[251] Alexandru Adrian Tole et al. Big data challenges. *Database systems journal*, 4(3):31–40, 2013.

[252] Mikalai Tsytsarau, Themis Palpanas, and Malu Castellanos. Dynamics of news events and social media reaction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 901–910, 2014.

[253] Mikalai Tsytsarau, Themis Palpanas, and Kerstin Denecke. Scalable detection of sentiment-based contradictions. *DiversiWeb, WWW*, 1:9–16, 2011.

[254] A. Tyler, L. Reuven, and C. Slava. chapter 1. In *Streaming Systems*. O'reilly Media, 2018.

[255] Balaji Varanasi and Sudha Belida. *Spring REST*. Apress, 2015.

[256] Massimo Villari, Antonio Celesti, Maria Fazio, and Antonio Puliafito. Alljoyn lambda: An architecture for the management of smart environments in iot. In *2014 International Conference on Smart Computing Workshops*, pages 9–14. IEEE, 2014.

[257] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.

[258] Mehul Nalin Vora. Hadoop-hbase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 1, pages 601–605. IEEE, 2011.

[259] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.

[260] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, 2015.

[261] Huijuan Wang, Yuan Yu, and Quanbo Yuan. Application of dijkstra algorithm in robot path-planning. In *2011 second international conference on mechanic automation and control engineering*, pages 1067–1069. IEEE, 2011.

[262] J. Wang, X. Peng, Z. Xing, K. Fu, and W. Zhao. Contextual recommendation of relevant program elements in an interactive feature location process. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 61–70.

[263] J. Wang, X. Peng, Z. Xing, K. Fu, and W. Zhao. Contextual recommendation of relevant program elements in an interactive feature location process. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 61–70.

[264] Qi Wang, Chenming Xu, Yangming Zhou, Tong Ruan, Daqi Gao, and Ping He. An attention-based bi-gru-capsnet model for hypernymy detection between compound entities. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1031–1035. IEEE, 2018.

[265] Shuai Wang, Zhiyuan Chen, and Bing Liu. Mining aspect-specific opinion using a holistic lifelong topic model. In *Proceedings of the 25th international conference on world wide web*, pages 167–176. International World Wide Web Conferences Steering Committee, 2016.

[266] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.

[267] Tom Wanyama. Static and dynamic coalition formation in group-choice decision making. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 45–56. Springer, 2007.

[268] R. Wei, S. Wu, D. Liu, Y. Zheng, S. Li, and R. Xu. Detection method of track locating point based on yolo v3. In *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 961–966, 2020.

[269] Moritz Weiten. Ontostudio® as a ontology engineering environment. In *Semantic knowledge management*, pages 51–60. Springer, 2009.

[270] Ward Whitt. Continuity of generalized semi-markov processes. *Mathematics of operations research*, 5(4):494–501, 1980.

[271] M. Winlaw, M. B. Hynes, A. Caterini, and H. D. Sterck. Algorithmic acceleration of parallel als for collaborative filtering: Speeding up distributed big data recommendation in spark. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 682–691.

[272] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 26(3):1–37, 2008.

[273] C. Xia, X. Jiang, Liu Sen, Luo Zhaobo, and Yu Zhang. Dynamic item-based recommendation algorithm with time decay. In *2010 Sixth International Conference on Natural Computation*, volume 1, pages 242–247.

[274] R. Xia, J. Jiang, and H. He. Distantly supervised lifelong learning for large-scale social media sentiment analysis. *IEEE Transactions on Affective Computing*, 8(4):480–491, Oct 2017.

[275] D. Xiang, Y. Wu, P. Shang, J. Jiang, J. Wu, and K. Yu. Rb-storm: Resource balance scheduling in apache storm. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 419–423.

[276] D. Xiang, Y. Wu, P. Shang, J. Jiang, J. Wu, and K. Yu. Rb-storm: Resource balance scheduling in apache storm. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 419–423.

[277] D. Xiang, Y. Wu, P. Shang, J. Jiang, J. Wu, and K. Yu. Rb-storm: Resource balance scheduling in apache storm. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 419–423.

[278] D. Xiang, Y. Wu, P. Shang, J. Jiang, J. Wu, and K. Yu. Rb-storm: Resource balance scheduling in apache storm. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 419–423, July 2017.

[279] F. Xie, M. Xu, and Z. Chen. Rbra: A simple and efficient rating-based recommender algorithm to cope with sparsity in recommender systems. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 306–311.

[280] F. Xie, M. Xu, and Z. Chen. Rbra: A simple and efficient rating-based recommender algorithm to cope with sparsity in recommender systems. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 306–311.

[281] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*, pages 1–6, 2013.

[282] Libo Xu, Xingsen Li, Chaoyi Pang, and Yan Guo. Simplified neutrosophic sets based on interval dependent degree for multi-criteria group decision-making problems. *Symmetry*, 10(11):640, 2018.

[283] Y. Yamato, H. Kumazaki, and Y. Fukumoto. Proposal of lambda architecture adoption for real time predictive maintenance. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 713–715.

[284] Y. Yamato, H. Kumazaki, and Y. Fukumoto. Proposal of lambda architecture adoption for real time predictive maintenance. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 713–715.

[285] Y. Yamato, H. Kumazaki, and Y. Fukumoto. Proposal of lambda architecture adoption for real time predictive maintenance. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 713–715.

[286] L. Yan, Z. Shuai, and C. Bo. Multi-sensor data fusion system based on apache storm. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 1094–1098, Dec 2017.

[287] Benjamin Z Yao, Xiong Yang, Liang Lin, Mun Wai Lee, and Song-Chun Zhu. I2t: Image parsing to text description. *Proceedings of the IEEE*, 98(8):1485–1508, 2010.

[288] L. Yao, Q. Z. Sheng, A. H. H. Ngu, J. Yu, and A. Segev. Unified collaborative and content-based web service recommendation. *IEEE Transactions on Services Computing*, 8(3):453–466, 2015.

[289] Xiaohan Yu and Zeshui Xu. Directed graph-based multi-agent coalitional decision making. *Knowledge-Based Systems*, 35:271–278, 2012.

[290] Xiaohan Yu and Zeshui Xu. Graph-based multi-agent decision making. *International Journal of Approximate Reasoning*, 53(4):502–512, 2012.

[291] Xiaohan Yu and Zeshui Xu. Graph-based multi-agent decision making. *International Journal of Approximate Reasoning*, 53(4):502–512, 2012.

[292] Amir Zadeh, Minghai Chen, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Tensor fusion network for multimodal sentiment analysis. *arXiv preprint arXiv:1707.07250*, 2017.

[293] Krista Rizman Žalik. An efficient kâĂš-means clustering algorithm. *Pattern Recognition Letters*, 29(9):1385–1391, 2008.

[294] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[295] Fan Zhang, Junwei Cao, Samee U Khan, Keqin Li, and Kai Hwang. A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications. *Future generation computer systems*, 43:149–160, 2015.

[296] Qing Zhang, Chaoyi Pang, Simon Mcbride, David Hansen, Charles Cheung, and Michael Steyn. Towards health data stream analytics. In *IEEE/ICME International Conference on Complex Medical Engineering*, pages 282–287, 2010.

[297] Hanqing Zhou, Amal Zouaq, and Diana Inkpen. Dbpedia entity type detection using entity embeddings and n-gram models. In *International Conference on Knowledge Engineering and the Semantic Web*, pages 309–322. Springer, 2017.

[298] Weiwei Zhuang, Qingshan Jiang, and Tengke Xiong. An intelligent anti-phishing strategy model for phishing website detection. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 51–56. IEEE, 2012.

[299] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[300] Theo Zschörnig, Robert Wehlitz, and Bogdan Franczyk. A personal analytics platform for the internet of things-implementing kappa architecture with microservice-based stream processing. In *International Conference on Enterprise Information Systems*, volume 2, pages 733–738. SCITEPRESS, 2017.